Coluna: Ask the Expert

Clube Delphi +PHP

DevMedia GROUP

Ano 7 · Edição 95 · R$9,90

Mini-Curso
Veja como criar um sistema on-line de controle para uma vídeo-locadora, com ASP.NET – Parte 3

Easy Delphi
Aprenda a criar e utilizar Stored Procedures em suas aplicações Win32 e .NET

Delphi for PHP
Como aplicar conceitos de POO em aplicações PHP - Parte 3

Coluna: QuickUpdate

Curso de ASP.NET 2.0 & RAD 2007

Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0

ISSN 1517990-7

NA WEB >>

Confira no portal ClubeDelphi PLUS 3 aulas sobre RAD 2007!

- **Desenvolvendo uma Aplicação Completa com PocketStudio**
  Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 4
- **Easy Delphi (Especial)**
  Aprenda como utilizar na prática recursos de um Web Services e como tirar proveito deles
- **PHP**
  Aprenda técnicas de Orientação a Objetos juntamente com o conceito MVC para criar aplicações em PHP

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

QuickUpdate

08 – Quick Update
Do estouro da bolha a Web 2.0
[*Adriano Santos e Guinther Pauli*]

Novidades
Mini-Curso
Web

10 – Curso de ASP.NET 2.0 e RAD 2007
Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0
[*Adriano Santos e Guinther Pauli*]

Mini-Curso
Tutorial

22 – Desenvolvendo uma Aplicação Completa com PocketStudio
Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 4
[*Ricardo C. Boaro*]

Web
Mini-Curso
Mão na Massa

30 – Controle on-line de vídeo-locadora - Parte 3
Veja como criar um sistema on-line de controle para uma vídeo-locadora
[*Maikel Marcelo Scheid*]

Mão na Massa

36 – Utilizando Threads na Prática
Veja todos os detalhes e dicas de como e onde utilizar Threads em suas aplicações
[*Ricardo C. Boaro*]

Easy

40 – Stored Procedures
Aprenda a criar e utilizar Stored Procedures em suas aplicações Win32 e .NET
[*Maikel Marcelo Scheid*]

Easy

48 – Consumindo Web Services
Aprenda como utilizar na prática recursos de um Web Services e como tirar proveito deles
[*Maikel Marcelo Scheid e Edinei Daniel Steffen*]

Delphi for PHP
Boas Práticas
Expert

56 – Orientação a Objetos no Delphi for PHP
Como aplicar conceitos de POO em aplicações PHP – Parte 3
[*Rodrigo Carreiro Mourão*]

PHP
Boas Práticas
Expert

62 – Programando em PHP usando MVC
Aprenda técnicas de Orientação a Objetos juntamente com o conceito MVC para criar aplicações em PHP
[*Daniel Ribeiro*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Delphi for PHP] Artigo com foco na versão PHP do Delphi

[Mini-Curso] Artigo que faz parte de um mini-curso

[QuickUpdate] Esse tipo de artigo 'atualiza' o leitor rapidamente sobre um determinado assunto, sem entrar muito a fundo.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado

[Easy] Com foco no desenvolvedor iniciante

[Mini-Curso] Artigo que faz parte de um mini-curso

[Tutorial] Artigo no estilo tutorial passo a passo

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre.

[PHP] Artigo com foco na linguagem PHP (não necessariamente utilizando o Delphi PHP)

[Boas Práticas] A Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas, mas também as melhores práticas – Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

ClubeDelphi

Ano 8 - 95ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Adolfo Sabino
simininu@yahoo.com.br

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Apoio
ASSESPRO PARANÁ

EDITORIAL

O .NET Framework 2.0, juntamente com o ASP.NET 2.0, foi lançado oficialmente no dia 7 de novembro de 2005, em San Francisco, CA. Estive no evento e conferi de perto as grandes e tão esperadas novidades para essa nova versão do framework Web de mais sucesso no mundo. Lembro-me que, no meio da apresentação, chamaram uma menina que era estudante (deveria ter uns 14 ou 15 anos) e ela mostrou na frente de quase 10 mil pessoas, em 5 minutos, como construir um BLOG completo com ASP.NET 2.0, demonstrando assim que qualquer pessoa pode, com a maior facilidade, criar Web Sites totalmente funcionais em minutos.

E isso é o ASP.NET 2.0, uma série de recursos comparados à versão anterior, 1.1. Resumindo os novos recursos temos: novos controles de dados, como GridView, DetailsView, FormView, um novo conceito de acesso a dados que "encapsula" o ADO.NET, chamado DataSources, além do ObjectDataSource que permite tratar classes de negócio como fonte de dados, Master Pages, novos controles para navegação (Wizard, TreeView, Menu, SiteMap), controles de Login que fazem praticamente todo o trabalho pesado de autenticar usuários em um site, o novo mecanismo de providers, Profiles, Web Parts, AJAX, Themes & Skins (você vai poder fazer com suas páginas Web algo semelhante ao que faz com os temas do Windows) entre outras muitas novidades.

E nesta edição, iniciamos um novo mini-curso que vai mostrar como utilizar os novos e principais recursos do ASP.NET 2.0 para criar um Web Site, usando a mais nova versão do Delphi, o RAD Studio 2007, que trouxe finalmente o suporte ao .NET Framework 2.0.

O Ricardo Boaro continua a sua série sobre o desenvolvimento de aplicações para o sistema PalmOS, com o IDE PocketStudio, bem semelhante à IDE do Delphi. O Maikel continua a construção da locadora on-line em ASP.NET, porém usando o framework 1.1 e Delphi 2006.

Na sessão Easy Delphi temos mais dois excelentes artigos do Maikel. Tenha uma bela introdução à utilização de Stored Procedures com Firebird, esse que considero um dos principais recursos de qualquer SGBD. No outro artigo, veja como consumir Web Services, essa tecnologia que já se tornou padrão para intercâmbio e comunicação de dados e serviços entre plataformas e sistemas heterogêneos.

Na sessão Delphi for PHP, o Rodrigo Mourão explica como utilizar Design Patterns com o Delphi for PHP, para criação de aplicações usando boas práticas de desenvolvimento OO. E na sessão PHP, o Daniel explica e aplica na prática o poderoso recurso de MVC (Model-View-Controller) em aplicações PHP.

Grande abraço e sucesso com o Delphi!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 580 Vídeo Aulas!
- 7 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: VTX535**

O acesso é válido por 30 dias a partir da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

Informativo ClubeDelph

Portal ClubeDelphi
www.clubedelphi.net/portal

+580 vídeo aulas e 7 cursos online

Brinde na web desta edição

3 Vídeos

RAD Studio 2007 – Instalando e conhecendo o RAD Studio 2007
Veja nessas vídeo-aulas de Luciano Pimenta todas as novidades do RAD Studio 2007. Além das novidades o Luciano mostra os detalhes de criação de uma aplicação ASP.NET 2.0 com suporte ao .NET Framework 2.0

RAD Studio 2007 Parte I
www.devmedia.com.br/articles/viewcomp.asp?comp=8356

RAD Studio 2007 Parte II
www.devmedia.com.br/articles/viewcomp.asp?comp=8357

RAD Studio 2007 Parte III
www.devmedia.com.br/articles/viewcomp.asp?comp=8392

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link: *www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:

Login: **DVM.CD** Senha: **VTX535**

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Copiando itens do ListBox para a memória

Mestres, como faço para copiar todos os itens de um *ListBox* para a memória e depois poder colar em outro *ListBox* ou outro controle qualquer?

Valéria

Valéria, precisaremos utilizar a unit *ClipBrd* que gerencia e ajuda a controlar a API do Windows referente ao *ClipBoard*. Declare essa *Unit* no seu *Uses* e inclua duas *procedures* como na **Listagem 1**. Para usá-las, inclua 2 *ListBoxes* e dois *Button's*. No primeiro *Button* digite o código a seguir para copiar os dados para a memória.

```
ListBoxToClipboard(ListBox1, 1024, True);
```

Em seguida digite o código a seguir no botão destinado a colar esses dados.

```
ClipboardToListBox(Listbox2);
```

Listagem 1. Procedures para copiar itens do ListBox

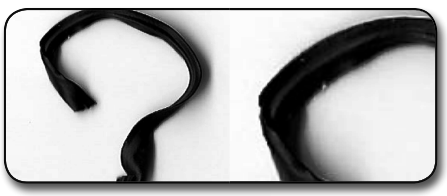
```
procedure TForm1.ListBoxToClipboard(ListBox: TListBox; BufferSize: Integer; CopyAll: Boolean);
var
  Buffer: PChar;
  Size: Integer;
  Ptr: PChar;
  I: Integer;
  Line: string[255];
  Count: Integer;
begin
  if not Assigned(ListBox) then
    Exit;
  GetMem(Buffer, BufferSize);
  Ptr   := Buffer;
  Count := 0;
  for I := 0 to ListBox.Items.Count - 1 do
  begin
    Line := ListBox.Items.strings[I];
    if not CopyAll and ListBox.MultiSelect and (not ListBox.Selected[I]) then
      Continue;
    Count := Count + Length(Line) + 3;
    if Count = BufferSize then
      Break;
    { Append to buffer }
    Move(Line[1], Ptr^, Length(Line));
    Ptr    := Ptr + Length(Line);
    Ptr[0] := #13;
    Ptr[1] := #10;
    Ptr    := Ptr + 2;
  end;
  Ptr[0] := #0;
  ClipBoard.SetTextBuf(Buffer);
  FreeMem(Buffer, BufferSize);
end;
procedure TForm1.ClipboardToListBox(ListBox: TListbox);
begin
  if not Assigned(ListBox) then
    Exit;
  if not Clipboard.HasFormat(CF_TEXT) then
    Exit;
  Listbox.Items.Text := Clipboard.AsText;
end;
```

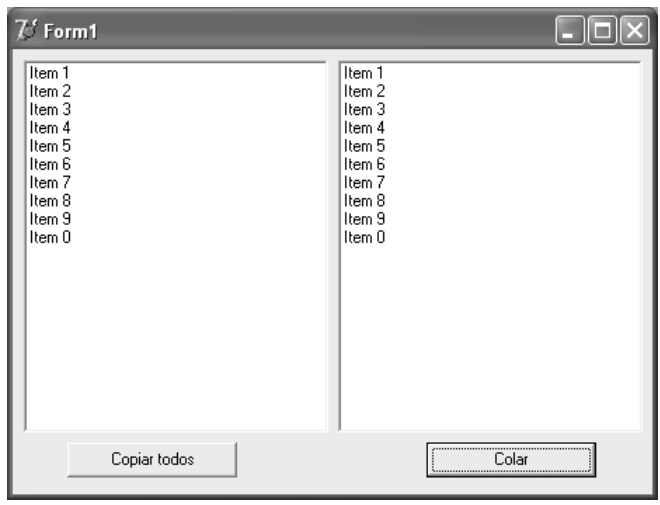
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback





Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO PROFISSIONAL 1

3 domínios independentes
50 GB de transferência
30 GB de e-mails com SSL
1 GB de espaço
Painel de controle
ASP, ASP.NET 3.5 E PHP 5
Access ilimitado
3 bancos MYSQL 5
1 banco SQL Server 2005 Express

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Nesta seção você encontra artigos que "atualizam" o leitor rapidamente sobre um determinado assunto.

Quick Update

Do estouro da bolha a Web 2.0

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

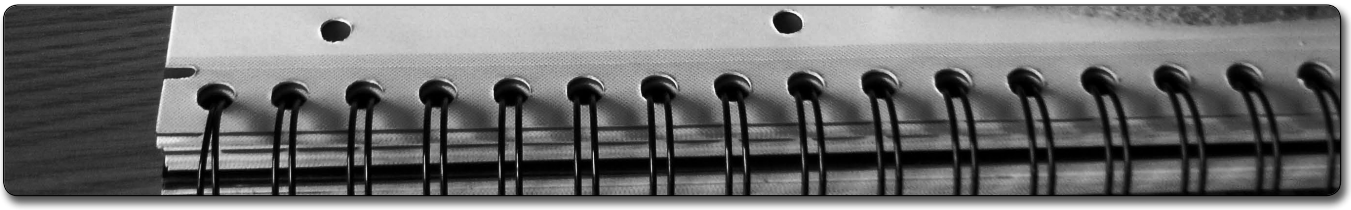
Guinther Pauli
(guinther@devmedia.com.br)
é Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Delphi Certified: 6, 7, 2005, 2006, 2007, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Trabalha com Delphi há 13 anos e .NET há 8 anos. Ministra treinamentos em ASP.NET 2.0 e 3.5, em RAD 2007 ou Visual Studio 2008.

Quem não se lembra do tão conhecido estouro da bolha em 1999/2000? Naquela época a internet estava começando a se tornar uma mania mundial. Quem sobreviveu ao estouro dessa bolha certamente lucrou bastante. O grande diferencial da época era baseado no conceito de e-commerce ou comércio eletrônico. Foi o comércio eletrônico quem moveu a internet e fez com que muitas empresas passassem a vender e comprar mais em vista da facilidade que a própria internet proporcionava.

Com o e-commerce em alta empresas fabricantes de softwares para desenvolvimento passaram a integrar em suas IDE's recursos necessários para o desenvolvimento rápido de aplicações Web. A Borland foi uma delas, integrando ao Delphi alguns recursos que, certamente, facilitariam e muito a criação de sites.

O WebBroker foi o primeiro passo para que o Delphi pudesse prover uma solução Web. O lançamento do Delphi 3 veio acompanhado do WebBroker e era possível desenvolver páginas Web com uma certa facilidade. Era possível até fazer a geração automática de códigos HTML através de componentes "Producers".

Ao desenvolvermos uma aplicação utilizando esse recurso, uma dll era compilada e nela ficava contida toda a regra de negócio de nossa aplicação. Essa dll era distribuída e alocada em pastas protegidas no servidor Web. O grande problema de se desenvolver aplicações Web com WebBroker se dava porque a dll era instanciada pelo primeiro internauta a acessar a página e ficava alocada em memória. Dessa forma o desenvolvedor precisava ter muita atenção ao criar variáveis e objetos, pois os mesmos não eram destruídos automaticamente. Outro grande problema é que não tínhamos como gerenciar sessões, recurso importantíssimo em páginas de comércio eletrônico. Outros dois recursos interessantes do WebBroker eram o



Request e *Response*. Com eles era possível, grosseiramente falando, receber e responder a ações da página.

Logo em seguida, nas versões posteriores ao Delphi 3, outra grande novidade foi incluída na IDE do Delphi: WebSnap (Delphi 6). O WebSnap foi o sucessor do WebBroker trazendo novidades como a modualização e suporte a sessões. Até hoje muitos desenvolvedores utilizando o WebSnap para criar suas aplicações Web. Com ele podemos integrar nossas aplicações com sites da Web ou desenvolvê-los. O mais importante é que ele possui suporte total aos mais diversos servidores Web, tais como Apache, Netscape e Microsoft Information Services (IIS). Acredito que o grande diferencial dele é poder conectar facilmente uma aplicação Win32 a um banco de dados on-line, ou seja, não necessariamente na mesma rede onde a aplicação roda, possibilitando o desvolvimento de sistemas capazes de funcionar em qualquer parte do planeta acessando uma única base de dados.

Enfim chegamos ao IntraWeb, um conjunto de componentes muito semelhantes aos da VCL do Delphi. A promessa para esses componentes era que o desenvolvedor poderia criar sites como se estivesse programando para Win32, ou seja, sem nenhum grande esforço. A "real" disso tudo é que o IntraWeb possui suas vantagens e desvantagens. Um grande vantagem é sua curva de aprendizado. Como ele é praticamente um Framework baseado na VCL do Delphi, seus componentes e a programação em si são parecidíssimos, facilitando o aprendizado. Mas se enganam aqueles que acham a melhor solução em Web. Aplicações geradas com IntraWeb, hoje chamado de *VCL for the Web* no RAD Studio 2007, são do tipo *state-full*. Isso é extremamente ruim, pois o site torna-se pesado e sem condições de aceitar múltiplas conexões simultâneas. Em outras palavras, um site desenvolvido com IntraWeb não suporta uma grande quantidade de usuários acessando-o ao mesmo tempo.

Web 2.0

Hoje em dia só se fala em Web 2.0, mas o que é exatamente a Web 2.0? A palavra chave para explicar a Web 2.0 é: integração. A internet está deixando de ser uma mera fonte de informações para se tornar um meio de comunicação. Com recursos cada vez mais avançados, a internet nos proporciona hoje ferramentas que antes não eram possíveis. A integração e resposta do usuário a determinadas ações são o ponto principal da nova cara da Web. A prova disso está na quantidade de comunidades, blogs, fóruns, canais de discussão, portais de jogos e entretenimento, planilhas eletrônicas e documentos on-line, disponíveis a todos. Enfim, podemos afirmar que estamos em uma nova era e para que essa nova era seja bem aproveitada precisamos novas linguagens e novas ferramentas. É aí que entram ASP.NET e PHP por exemplo, que atualmente estão em alta na comunidades de desenvolvimento, aliados claro ao todo poderoso AJAX. ●

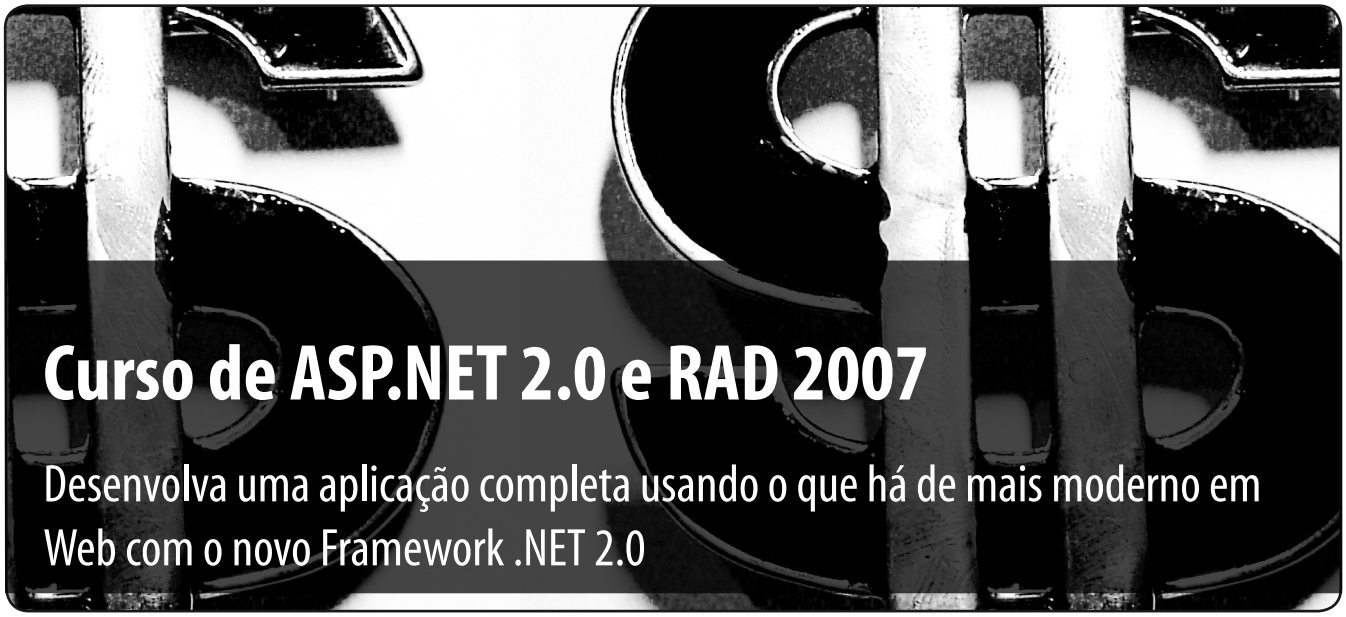
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET



Curso de ASP.NET 2.0 e RAD 2007

Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0

Guinther Pauli
(guinther@devmedia.com.br)
é Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Delphi Certified: 6, 7, 2005, 2006, 2007, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Trabalha com Delphi há 13 anos e .NET há 8 anos. Ministra treinamentos em ASP.NET 2.0 e 3.5, em RAD 2007 ou Visual Studio 2008.

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revistas ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Bem-vindos a mais uma inovação da revista ClubeDelphi: um curso para você conhecer os principais recursos do ASP.NET 2.0 utilizando o RAD Studio 2007, a mais nova versão do Delphi. Como ferramenta de banco de dados utilizaremos o Microsoft SQL Server 2005 Express Edition, que é gratuito. Nesse minicurso, você conhecerá as principais novidades para o desenvolvimento de aplicações trazidas pelo ASP.NET 2.0, que facilitarão o desenvolvimento de Web Sites, gerando grande produtividade para você desenvolvedor.

Vamos mostrar os principais novos controles, acesso a base de dados, mais dicas e truques que serão úteis para a construção e organização de Web Sites. Baseando-se em uma aplicação comercial simples, os recursos serão demonstrados de forma que se aproxime ao máximo com a rotina de um desenvolvedor.

O cenário

Neste curso, criaremos um sistema de controle de orçamentos on-line, com os seguintes requisitos:

- Cadastro de categorias;
- Cadastro de produtos;
- Solicitação de orçamento;
- Controle completo de segurança – cadastro de grupos, usuários e permissões de acesso.

Antes de começar...

Antes de codificar qualquer coisa em ASP.NET, é importante que você tenha em mente alguns conceitos e tome alguns cuidados importantes, principalmente se está migrando de uma solução Desktop/VCL:

- Não queira fazer na Web as coisas exatamente como você faz em aplicações Windows/Desktop - o uso de *Web Forms* do ASP.NET ajuda, claro, mas saiba que você está rompendo de uma vez duas barreiras: mudança de framework (VCL para .NET

FCL) e mudança de paradigma (Windows para Web). Um browser não é um formulário Delphi, infelizmente, então se acostume às limitações do HTML e HTTP;

• Saiba como funciona o modelo de execução do ASP.NET – é importantíssimo lembrar que o ASP.NET (e aplicações *Web* como um todo) são *stateless*: cada requisição é processada como se fosse a primeira. Além disso, o ASP.NET cria e recria o formulário no servidor para cada requisição, o que é bem diferente do modelo "TForm-based". É claro, temos alguns recursos para tratar sessão, cache, *ViewState* etc.

Download e instalação do MS SQL Server 2005 Express

Acesse o endereço *www.microsoft.com/express/sql/download/default.aspx* e localize na coluna da esquerda os itens *Install Microsoft SQL Server 2005 Express Edition* e *SQL Server Management Studio Express*. Abaixo de cada um existe uma pequena descrição e em seguida um link para download. Trata-se do próprio banco de dados e seu gerenciador, respectivamente. Precisaremos de ambos arquivos para administrar nossa base de dados.

Após o download teremos que executar os dois arquivos que são:

• *SQLEXPR32_PTB.exe*: Arquivo de instalação do banco de dados propriamente dito;

• *SQLServer2005_SSMSEE.msi*: Arquivo de instalação do SQL Management Studio Express, o gerenciador do banco (front-end, semelhante ao que é o IBExpert para o Firebird).

Durante a instalação teremos que configurar diversas telas, porém daremos atenção somente as mais importantes. Após carregar o arquivo de instalação, o MS SQL Server Express faz a instalação automática dos componentes *Microsoft SQL Native Cliente* e *Arquivos de Suporte à Instalação do Microsoft SQL Server 2005* (**Figura 1**).

Após aceitar os termos de licença de uso, o programa de instalação solicita informação para registro do software. Apesar de gratuito, o MS SQL Server 2005 Express solicita um nome e empresa para registro no seu banco de dados. Apenas clique em avançar e logo em seguida escolha os componentes que serão instalados. Na tela seguinte (**Figura 2**) escolha o modo

Figura 1. Instalação do MS SQL Server 2005 Express

Figura 2. Modo de autenticação do MS SQL Server 2005 Express

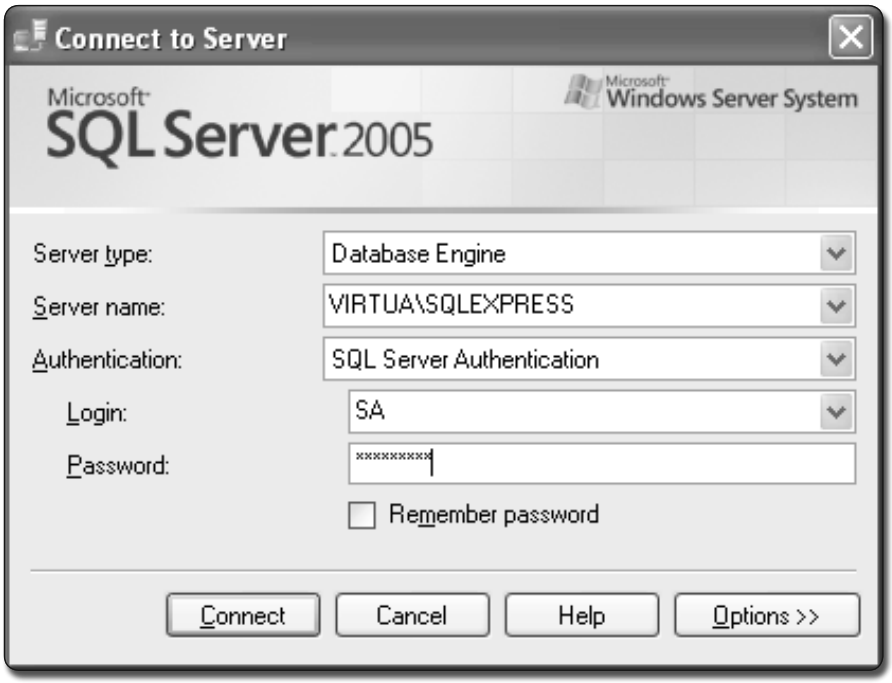


Figura 3. Tela de login do gerenciador

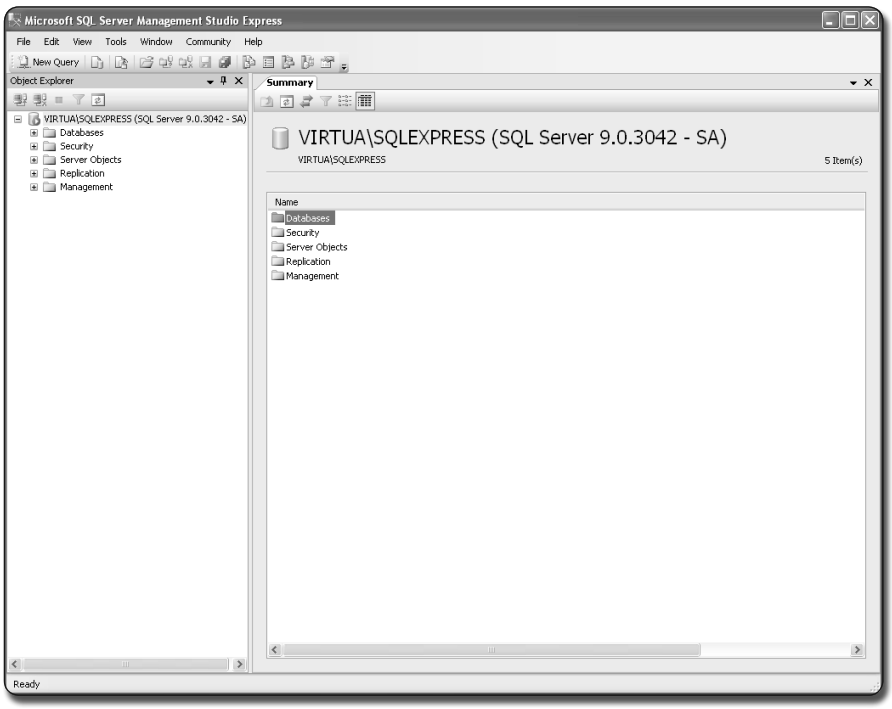


Figura 4. Management Studio Express

de autenticação com o banco de dados. Escolha a opção dois, *Modo Misto* e indique uma senha. Por fim clique em avançar até iniciar a instalação dos componentes.

Feita a instalação do banco de dados, inicie a instalação do gerenciador que não tem segredos. Apenas acompanhe o assistente e aguarde o término.

> **Nota:** Por que usar o SQL Server Express? Simples, pois esse é o melhor banco para se trabalhar com ASP.NET. A performance é muito maior comparada a qualquer outro banco, no caso de aplicações Web. E rapidez na Web é um fator crítico, lembre-se.

Criando o banco de dados

Para iniciar o projeto, vamos primeiramente criar o banco de dados e as tabelas necessárias para cada tela de cadastro. As tabelas são: *Categorias e Produtos*. Neste artigo, criaremos o banco de dados utilizando a ferramenta mencionada anteriormente. Por isso acesse o menu *Iniciar|Programas>Microsoft SQL Server 2005> SQL Server Management Studio Express*. Uma janela como na **Figura 3** será mostrada ao entrar no gerenciador.

Em *Server name* digite o nome do servidor, que normalmente é o nome da MAQUINA\SQLEXPRESS (**Figura 3**), e em seguida selecione *SQL Server Authentication* em *Authentication*. Digite o login ("SA") e a senha. Feito isso o *Microsoft SQL Server Management Studio Express* será aberto com uma aparência semelhante à **Figura 4**.

Para criar um novo banco de dados, expanda o item *DataBases* e em seguida clique com o botão direito nele. Escolha a opção *New DataBase* e dê o nome de "Propostas". Para criarmos as tabelas basta expandirmos o nosso banco de dados criado e clicar com o botão direito em *Tables* selecionando *New Table*. Em seguida uma janela será aberta e agora precisaremos apenas incluir os campos e seus atributos necessários. Vamos criar junto a tabela de *Categorias*.

Nas propriedades da tabela, em *Name*, digite o nome da tabela (**Figura 5**). Em seguida digite o nome *CATEGORIAID* em *Column Name*. Preencha o *Data Type*,

marque ou desmarque o item *Allow Nulls* conforme a necessidade. No campo *CATEGORIAID* criaremos um *Identify*, que é semelhante ao *Generator* do Firebird, ou seja, gera uma numeração para o campo automaticamente. Expanda o item *Identify Specification* e marque o atributo (*Is Identify)* como *True*. Note que o sistema já atualiza automaticamente o campo *Indetify Increment* (para incrementar o campo). Por fim clique em *Ok* para finalizar a criação da tabela. Nossa primeira tabela deverá se parecer com a **Figura 6**.

Repita os mesmos passos anteriores e crie a tabela *Produtos* no sistema. Lembrando que o campo *PRODUTOID* será um *Identify*, assim como na tabela anterior. Crie a tabela como mostrado na **Figura 7**.

Após a execução de todos os passos teremos nosso banco de dados, tabelas, índices e tudo que é necessário para que nosso *BD* funcione perfeitamente.

> **Nota**: Nosso banco de dados possui somente duas tabelas, já que o orçamento é montado de forma on-line não trazendo a necessidade do armazenamento dos dados.

Criação da aplicação

Abra o RAD Studio 2007 e clique em *File>New>ASP.NET Web Application - Delphi for .NET*, dê o *Name* de "Propostas", aponte o caminho do aplicativo para *C:\Inetpub\wwwroot\Propostas* onde serão salvos os arquivos do projeto. Por fim selecione o servidor de aplicação, que pode ser *Internet Information Server* ou *Cassini Web Server*. Se preferir observe os dados na **Figura 8**.

Após a criação do projeto a ferramenta criará os arquivos iniciais. Além do arquivo *Default.aspx*, será possível visualizar também o arquivo *Web.config* que possui as principais configurações de nosso website.

Criando a estrutura da aplicação

A maioria das aplicações possui uma aparência padrão e estrutura semelhante de navegação/visualização. Normalmente existe um topo e uma lateral com menus ou opções que estarão disponíveis em todas as páginas. No ASP.NET 1.x, usávamos *User Controls* para criar áreas comuns nas aplicações Web. Apesar dos *User Controls* não terem sido criados especificamente para essa finalidade, resolviam grande parte dos problemas.

O ASP.NET 2.0 trouxe um recurso muito interessante chamado *Master Pages*, que permite a criação de diversos *templates* que serão usados como base para todas as páginas. Sendo assim, iniciaremos criando a *Master Page* que será usada em todo o mini-curso.

No RAD Studio selecione *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Master Page*. Uma caixa de diálogo solicitará o nome da página mestre, então digite "MiniCurso.master". Confirme com *Ok* para efetuar a criação da *Master Page*.

Ao criar a *Master Page*, a ferramenta apresenta um quadro contendo um objeto *ContentPlaceHolder*, que representa a seção onde ficará o conteúdo específico de cada página criada futuramente. Tudo o que estiver fora desse objeto será exibido em todas as páginas que fizerem uso dessa *Master Page*, e é nessa área onde normalmente coloca-se o logo da empresa, um menu de navegação, informações do usuário logado, e outras informações que são necessárias em todas as páginas do Web Site.

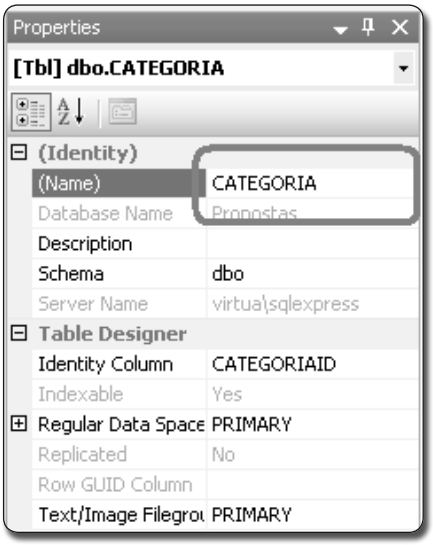


Figura 5. Nomeando a tabela

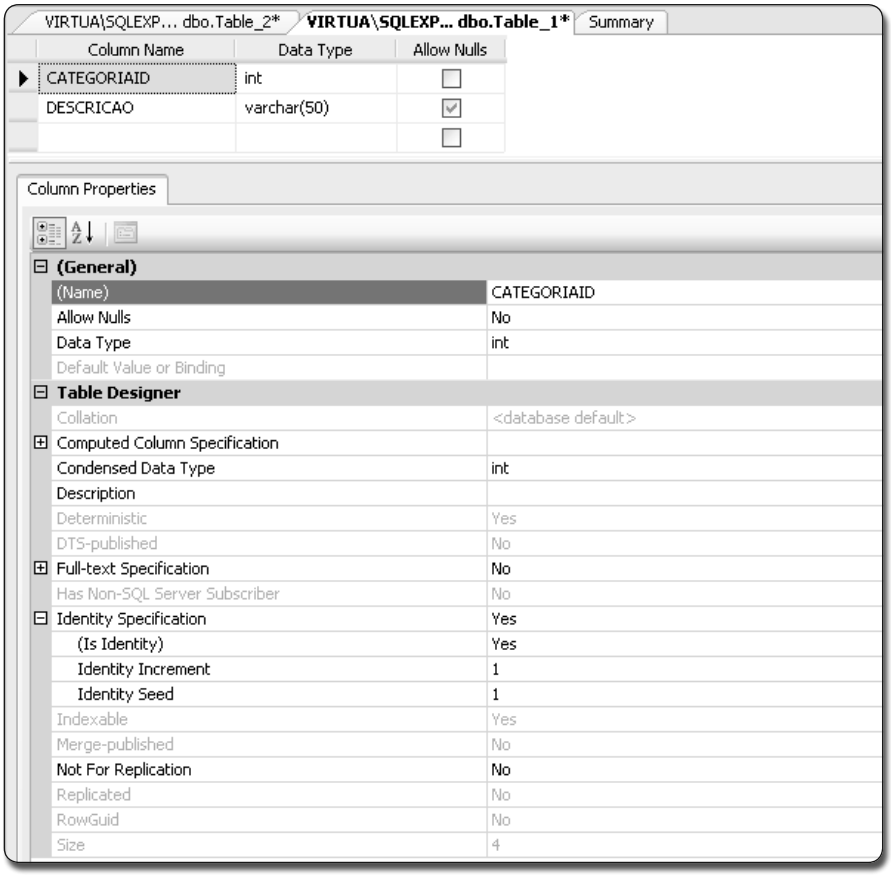


Figura 6. Criando uma tabela no MS SQL Server 2005 Express

Configure o layout da forma que achar mais conveniente, pois iremos incrementando a *Master Pages* com outros recursos importantes do ASP.NET. Após ter configurado a *Master Page*, criaremos a primeira página, que será o *Menu Principal* da aplicação, contendo links para as diversas funcionalidades do sistema. Para isso, clique em *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Content Page* e salve-o como "MenuPrincipal.aspx". Note que o segundo campo que aparece é justamente a seleção do arquivo *Master Page* (**Figura 9**).

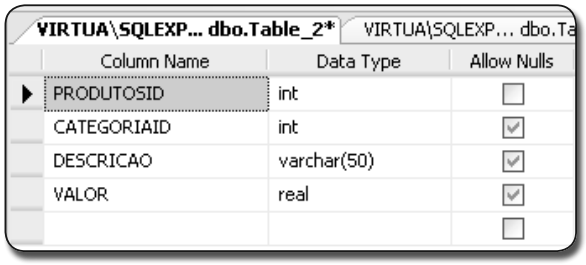
Após a criação do *Menu Principal* é possível visualizar em *Design Mode* uma tela semelhante à **Figura 10,** mostrando na parte superior o conteúdo da *Master Page* e na seção inferior, o objeto *Content* que armazenará todo o conteúdo da página. Se em algum momento for necessário alterar a *Master Page*, basta clicar com o botão direito na área da *Master Page*, selecionar a opção *Edit Master* e você será redirecionado para o arquivo, podendo assim efetuar as mudanças necessárias.

Navegação

A navegação de uma aplicação Web é algo muito importante, e no ASP.NET 2.0 juntamente com o RAD Studio 2007 temos ferramentas que facilitam muito essa tarefa. Foi criado um novo conceito chamado *Site Map*, onde, através de um arquivo XML, definimos todas as páginas do projeto conforme a hierarquia da aplicação. A partir desse recurso podemos tirar proveito de controles que utilizam como fonte de dados o arquivo *Web.sitemap*.

Para adicionar o arquivo, selecione *File|New>Other>Delphi .NET Projects>New ASP.NET Files>Site Map*. O arquivo XML é aberto e já traz o *template* básico para alteração. Arquivos XML trabalham de forma hierárquica, onde os nós definem quem será o pai e filho na relação, com isso adicionamos como nó principal do arquivo a página principal do sistema. A partir do nó principal a hierarquia segue até um nível infinito de pais/filhos. Altere o arquivo conforme a **Listagem 1**.

Após a definição da hierarquia, vamos



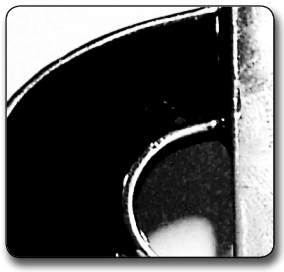


Figura 7. Criação da tabela Produtos

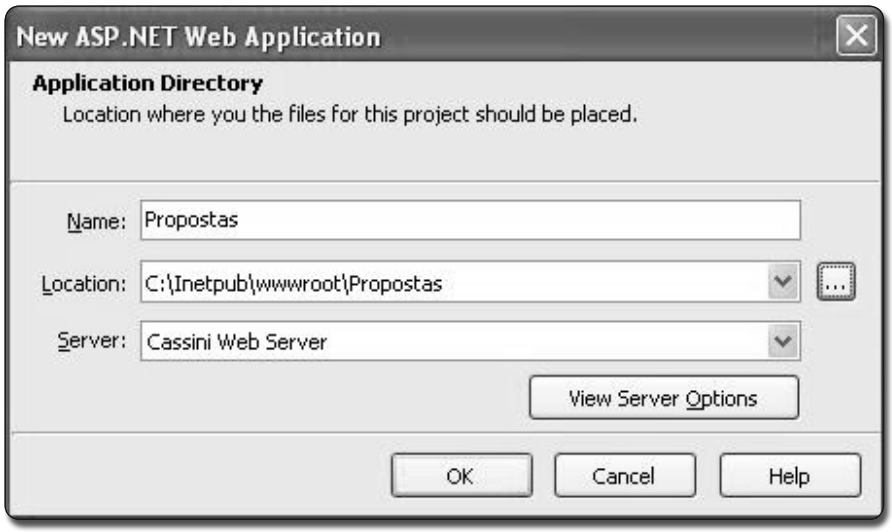


Figura 8. Criação do novo projeto do tipo Web Site

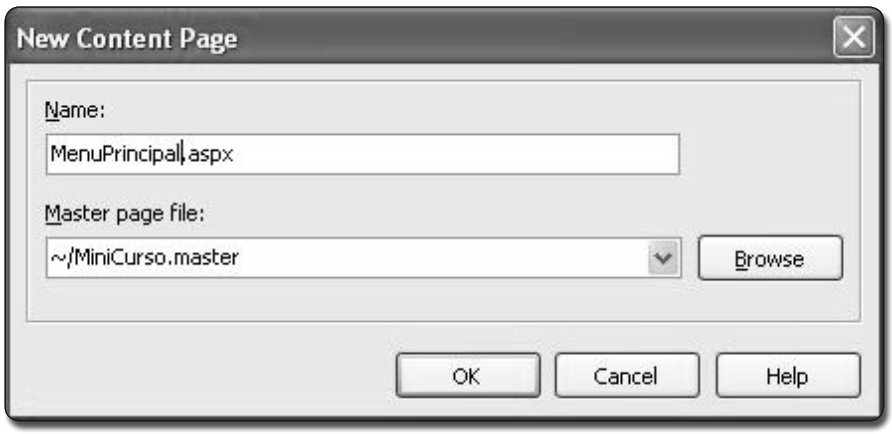


Figura 9. Criando página de conteúdo e selecionando o Master Page

Listagem 1. Mapa do site em arquivo XML

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode url="MenuPrincipal.aspx"
    title="Página Principal do Sistema"  description="">
   <siteMapNode title="Cadastros"
      description="Cadastros do Sistema">
        <siteMapNode url="Cadastro/Categoria.aspx"/>
        <siteMapNode url="Cadastro/Produto.aspx"/>
   </siteMapNode>
   <siteMapNode title="Orçamento"
      description="Solicitação de Orçamento" url="Orcamento/Orcamento.aspx"/>
   <siteMapNode title="Consulta"
      description="Consulta Geral" url="Consulta/Consulta.aspx"/>
  </siteMapNode>
</siteMap>
```

conhecer os controles que nos beneficiam e muito na navegação das páginas. Como a navegação ocorre em todas as páginas, vamos alterar os controles dentro da *Master Page*, assim todas as páginas passam a ter a navegação pronta. Abra em modo *Design* a *Master Page* já criada, e logo abaixo do texto de título adicione o controle *Menu* (**Figura 11)**, que está localizado na paleta na categoria *Web Navigation*.

Selecione o controle menu e altere as propriedades *Orientation* para *Horizontal*, com isso o menu ficará alinhado com o título. Modifique também a propriedade *StaticDisplayLevels* para 2, para iniciar a exibição a partir do nó 2 na hierarquia do arquivo *Site Map*. Abra a *Smart Tasks* (**Figura 12**) do menu, selecione *Auto Format* e altere o modelo conforme o seu gosto. Novamente na *Smart Tasks*, selecione *Choose Data Source>New Data Source*, selecione *Site Map* e clique em *OK* (**Figura 13**).

> **Nota**: Os novos componentes do ASP.NET que você vê na paleta não são nativos do Delphi, obviamente. Eles fazem parte do .NET Framework 2.0. O mesmo vale para os recursos utilizados, como MasterPages, SiteMap etc.

O menu está pronto, note que os valores contidos no arquivo XML já são exibidos**.** E outro controle muito interessante é o *SiteMapPath*, da aba *Web Navigation*, que exibe o "caminho" em que o usuário se encontra dentro do site. Adicione-o logo abaixo do menu. A única customização necessária é a do *template*, que pode ser feita a partir da *Smart Tasks*. Pronto, esse é um grande recurso muito interessante e simples de ser utilizado, e que enriquece muito qualquer aplicação.

Acessando o banco de dados

Aqui apresentaremos o cadastro de produtos e das categorias de produtos. Você verá a criação de dois cadastros completos, sem a necessidade de nenhuma linha de código, graças ao ASP.NET 2.0!

Vamos iniciar a criação das pastas da aplicação para o desenvolvimento. Serão criadas três pastas: "Cadastro", "Orcamento" e "Consulta", que receberão as políticas de segurança para acesso.

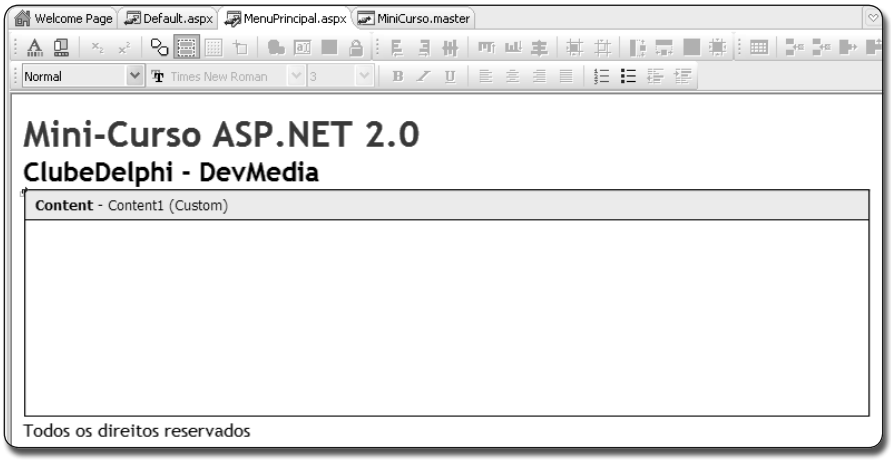


Figura 10. Utilização de Master Pages no menu principal do Web Site

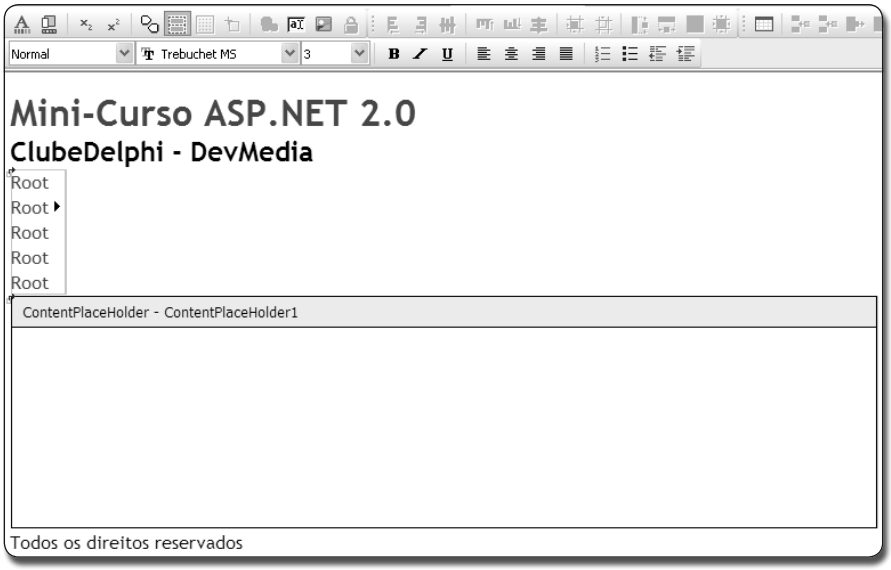


Figura 11. Menu do Sistema

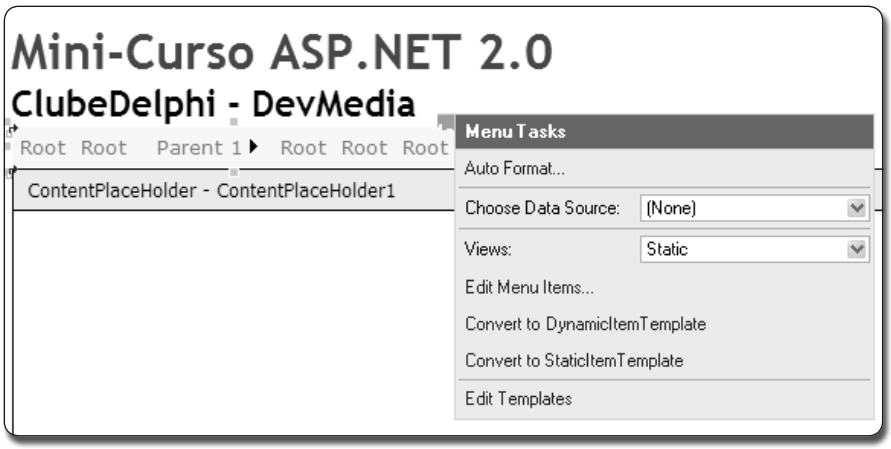


Figura 12. Seleção da Smart Tasks

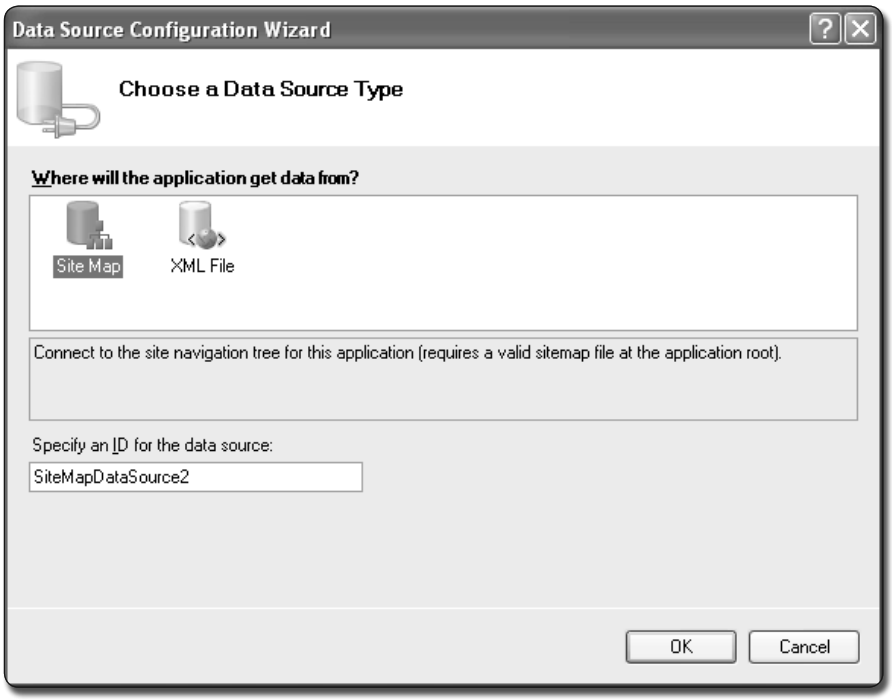


Figura 13. Seleção do Site Map

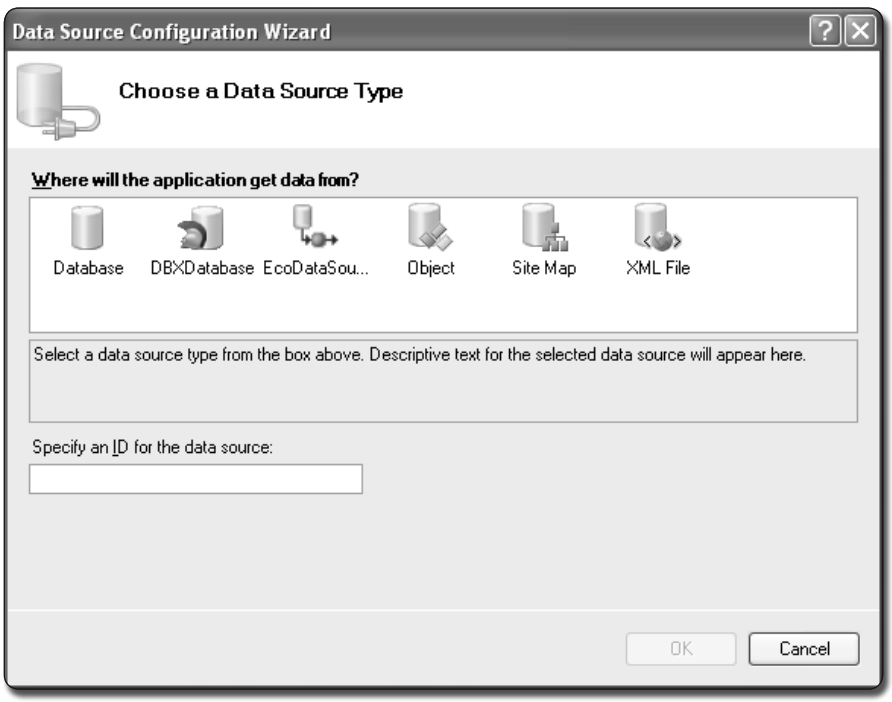


Figura 14. Selecionando o DataSource

Para iniciar a criação das pastas, clique com o botão direito no projeto e aponte para *New Folder*. Uma nova pasta é criada, nomeie-a para "Cadastro", repita a mesma operação para as pastas citadas anteriormente.

Cadastro de Categorias

Uma vez que as pastas já estão criadas, iniciaremos a construção da página de cadastro de categorias. Clique com o botão direito na pasta *Cadastro* e selecione a opção *Add New|Other*, aponte para o *template ASP.NET Content Page* e digite "Categoria.aspx" no campo *Name*.

Após a criação do *Categoria.aspx*, desenharemos a tela de cadastro com os campos já definidos na base de dados. Para isso, utilizaremos os novos controles de acesso a dados disponibilizados pelo ASP.NET 2.0 para facilitar o desenvolvimento. Um desses controles é o *DetailsView,* arraste-o a partir da paleta de componentes *Web Data* para dentro do objeto *Content*.

O *DetailsView* é utilizado para criar formulários com as operações básicas: inclusão, alteração e exclusão de registros. O layout do formulário padrão criado pelo controle pode ser alterado, mantendo a mesma funcionalidade das operações com o banco de dados. Após arrastar o controle, é possível definir uma formatação padrão atravéz da opção *Auto Format* da *Smart Tasks*.

Faremos o acesso ao banco através da opção *Choose Data Source* presente na

Nota do DevMan

Para que o DetailsView tenha uma aparência mais profissional e agradável para o usuário final, faremos algumas configurações nele, como a tradução de seus Label's. Dê um clique de direita no DetailsView e escolha Smart Tasks, em seguida selecione Edit Fields. Na parte inferior esquerda da janela você encontrará os campos usados no componente ("Selected Fields"). Clique em CategorialD e observe que em BoundField Properties que podemos modificar uma série de propriedades, como por exemplo Header text.

Outro segredo é que podemos traduzir os textos do controle como dito anteriormente. Para isso selecione o item CommandField. Encontre e traduza todos os itens, como por exemplo, os atributos: CancelText, DeleteText, EditText etc. Confirme a caixa de diálogo e note a diferença.

Smart Tasks. Selecione *New Data Source* e uma nova janela de configuração é aberta (**Figura 14**).

Escolha então *DataBase*, digite "SqlCategoria" em *Specify an ID for the data source* e clique em *OK*. Como essa conexão com o banco de dados será a primeira no projeto criaremos uma nova conexão, no RAD Studio, portanto clique em *New Connection*. Na tela seguinte digite, em *Server Name*, *MAQUINA\SQLEXPRESS* que é o nome do servidor. Em seguida mude para *Use Sql Server Authentication*, digite seu usuário e senha. Em *Select or enter a database name*, selecione *Propostas* e confirme (**Figura 15**).

Então clique em *Ok* e *Next* duas vezes, pois não faremos configurações nas telas seguintes. A última janela é aberta, onde selecionamos o objeto do banco de dados que vamos utilizar, nesse caso selecione a tabela *Categoria* (**Figura 16**) e por fim marque o "*" para que possamos trazer todos os dados da tabela. Clique no botão *Advanced*, que abrirá a janela da **Figura 17** onde podemos marcar a primeira opção, fazendo com que sejam geradas as instruções SQL para inserção, atualização e exclusão de dados.

Com o *DataSource* configurado, podemos utilizar os recursos do *DetailsView*. Selecione a *Smart Tasks* do *DetailsView* já adicionado e selecione *Enable Inserting* e se preferir *Enable Editing* e *Enable Deleting* (**Figura 18**).

Para o nosso exemplo, só utilizaremos o *DetailsView* para adicionar novas categorias, mas poderíamos utilizar também o *Update* e *Delete* (respectivamente as opções *Enable Editing* e *Enable Deleting*).

Selecione o controle e altere a propriedade *Default Mode* para *Insert*. Com isso o controle já é exibido na vizualização de inserção de dados e está pronto para uso. Abaixo do formulário vamos adicionar um *GridView*.

> **Nota**: O GridView é o sucessor do antigo DataGrid, muito mais aprimorado e com mais recursos. Utilize-o sempre a partir de agora.

Selecione na paleta de componentes o *GridView*, localizado na categoria *Web*

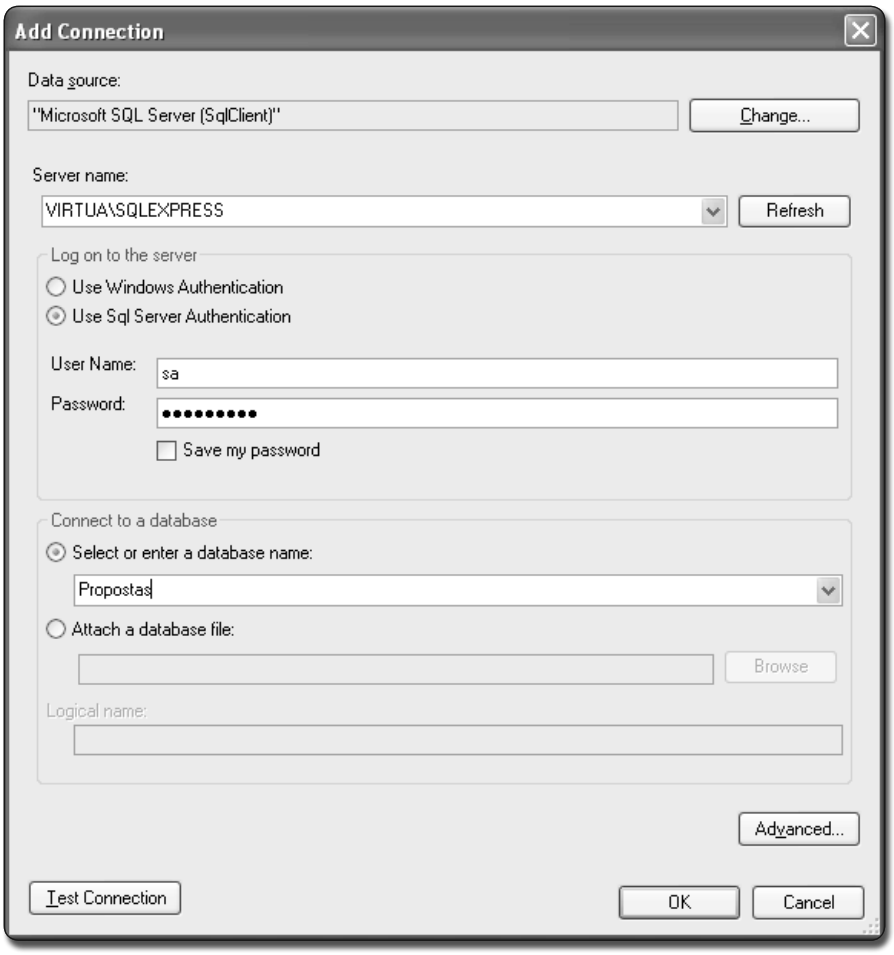


Figura 15. Selecionando o banco Propostas

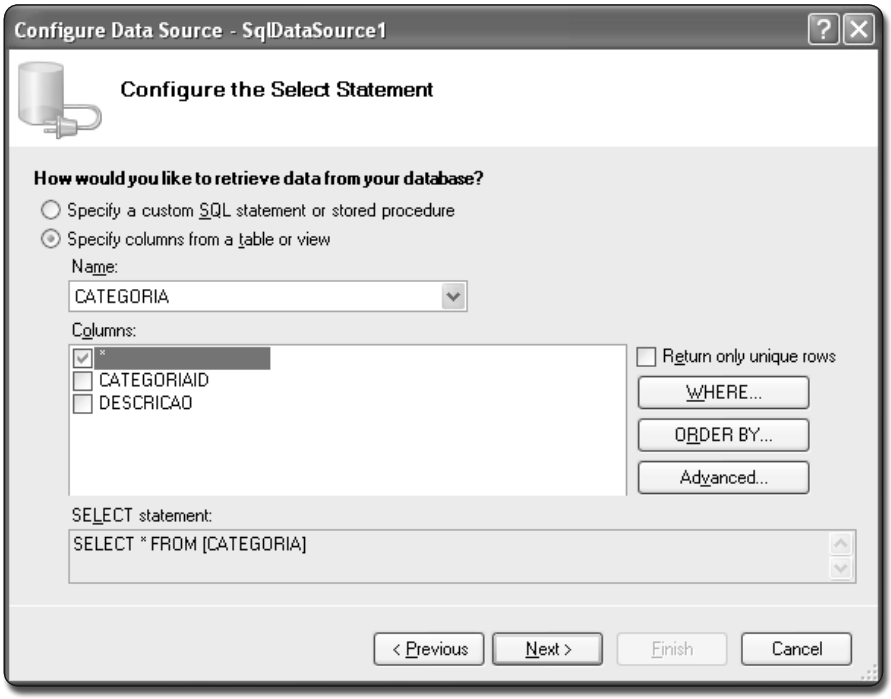
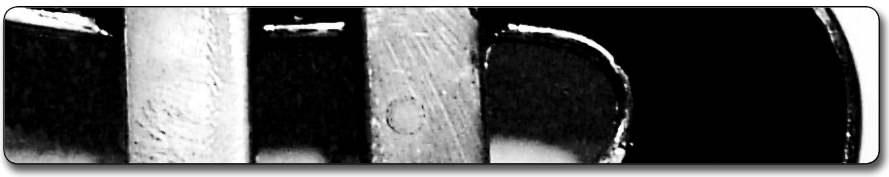


Figura 16. Selecionando a tabela Categoria do banco de dados



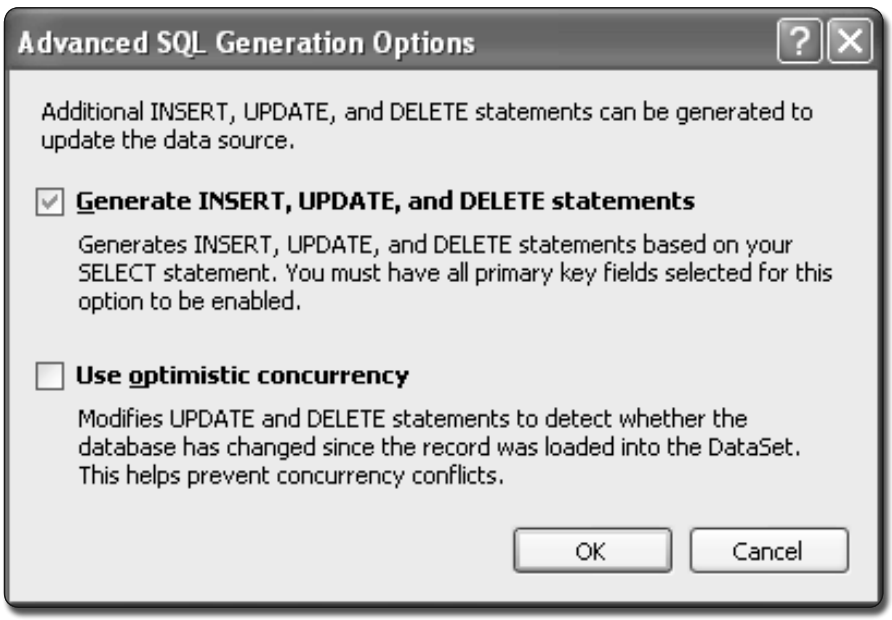


Figura 17. Gerando as instruções SQL automaticamente

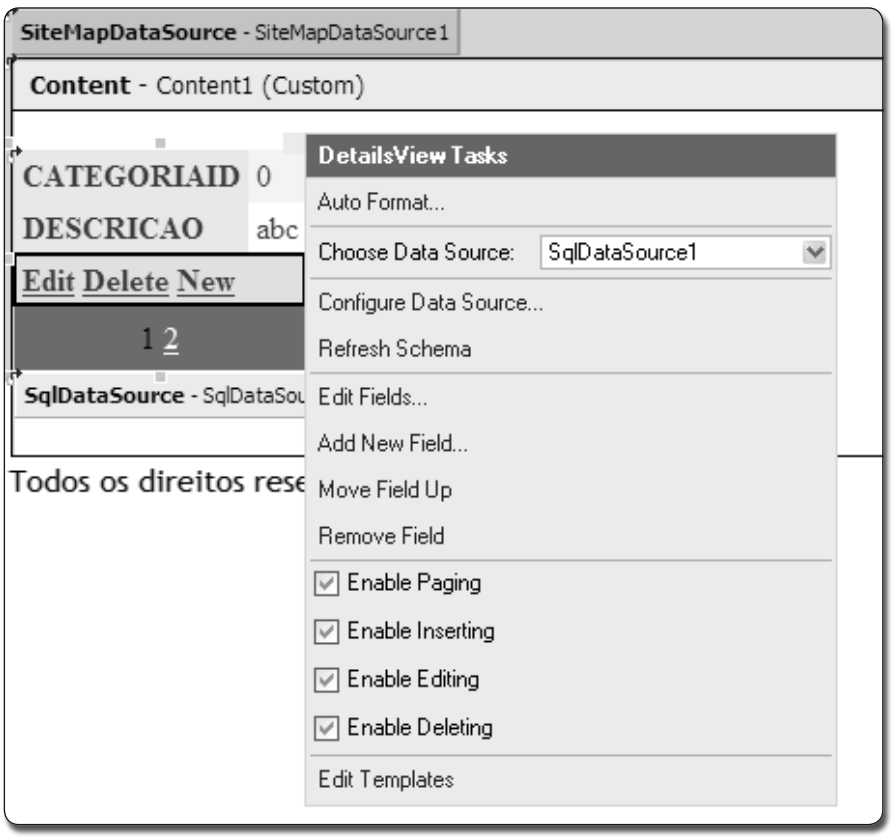


Figura 18. Configurando o controle DetailsView para inserção de dados

Data. Como já criamos uma *SqlDataSource* da tabela *Categoria*, podemos compartilhar a utilização do *SqlDataSource* com o *GridView*. Na *Smart Tasks* do *grid*, selecione em *Choose Data Source* o *SqlCategoria*.

Na própria *Smart Tasks* habilite as opções: *Enable Paging*, *Enable Sorting*, *Enable Editing* e *Enable Deleting*, sendo assim o controle está pronto para editar, deletar, ordenar valores e paginar. Para testar, no *Project Explorer* clique com o botão direito do mouse sobre a página e selecione *View in Browser*. Cadastre algumas categorias a seu gosto.

Cadastro de Produtos

O cadastro de produtos segue a mesma característica de layout do cadastro de categorias, somente com o *DetailsView* e *GridView*. Adicione ao projeto uma nova página chamada "Produto.aspx" vinculando à *MasterPage* já criada (semelhante a técnica utilizada no *página* anterior).

Adicione ao formulário o *DetailsView* e vamos configurar o *DataSource* de produtos. Na *Smart Tasks* do controle, escolha a opção *Choose Data Source* e

Figura 19. Configurando as colunas templates do DetailsView

selecione *New Data Source*. Novamente a janela de configuração do *DataSource* é aberta, selecione *DataBase*, digite "SqlProduto" para o *ID* e clique em *OK*.

Na escolha da conexão com o banco de dados, podemos escolher a *string* de conexão, selecionando a última chave incluída no arquivo *web.config*. Selecione *Propostas* e clique em *Next*. Escolha a tabela *Produto* e clique em *Advanced*. Na janela *Advanced*, marque a opção para gerar as instruções SQL de inserção, atualização e exclusão. Clique em *OK* e siga o restante das telas até o final. Novamente na *Smart Tasks*, selecione *Enabled Inserting*. Altere a propriedade *DefaultMode* para *Insert*.

Nesse caso temos uma nova necessidade, para o campo *CategoriaID* é criado um *TextBox*, porém a necessidade é de um *DropDownList* vinculada com a tabela *Categoria*, pois o usuário não precisará digitar o código da categoria, mas sim, apenas escolher a mesma. Para isso, selecione na *Smart Tasks* a opção *Edit Fields*. Em *Selected fields* selecione a coluna de categoria ("CategoriaID") e clique na opção *Convert this field into a TemplateField*.

Com isso temos uma coluna com os templates editáveis, e para esse cenário, vamos editar a coluna no template de *Insert*, denominado *InsertItemTemplate*. Clique em *OK* para fechar o editor. Dê um clique de direita sobre o *DetailsView*, escolha a opção *Field[1] - CategoriaID*. Será aberto o editor para adição de controles em "campos" templates (**Figura 19**).

Neste momento temos o *GridView* exibido com todos os tipos de templates. Vamos alterar somente o *InsertItemTemplate*, que é a exibição no momento em que o controle está em estado de inserção.

Delete o *TextBox* que está no *InsertItemTemplate* e em seu lugar, adicione um *DropDownList*. A primeira tarefa é o preenchimento do controle com todas as categorias existentes, para isso, selecione a *Smart Tasks* e em *Choose DataSource* escolha *New Data Source*. Selecione *DataBase* e clique em *OK*. Selecione novamente a *string* de conexão e os campos na tabela *Categoria*. Por fim clique em *Finish* (neste caso, não precisamos adicionar as instruções SQL de inserção, atualização e exclusão). Você será retornado novamente para o editor do *DropDownList*.

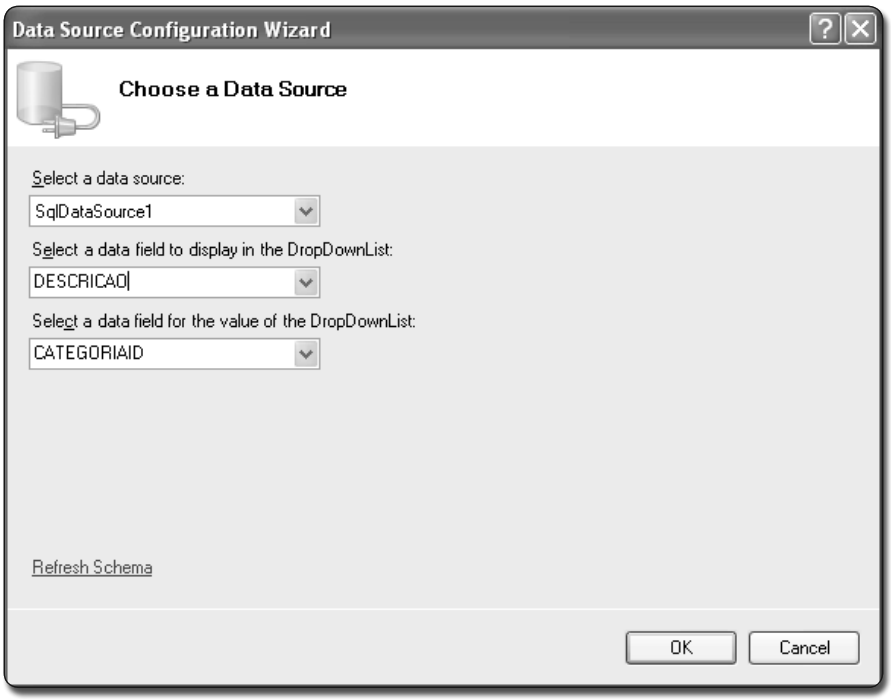


Figura 20. Selecionando os campos de exibição e coluna chave no DropDownList

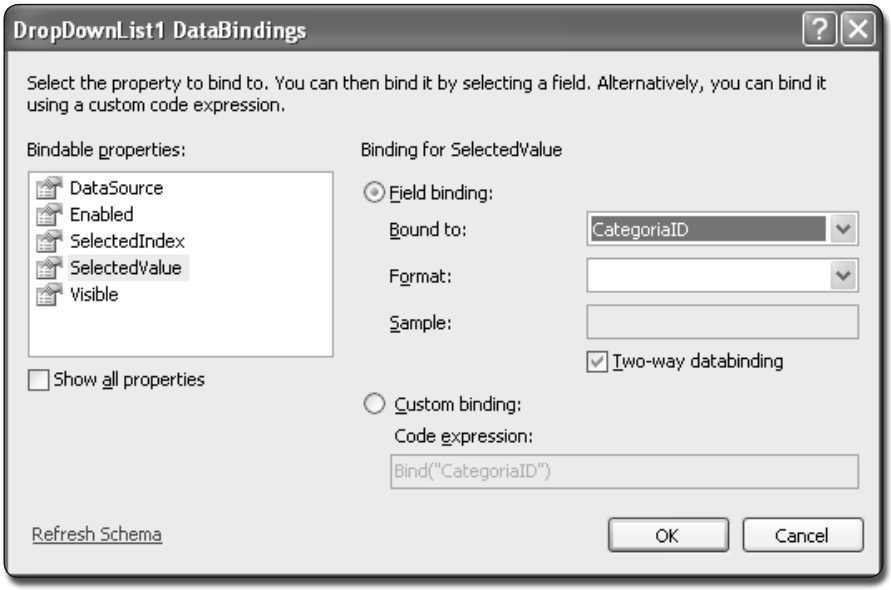
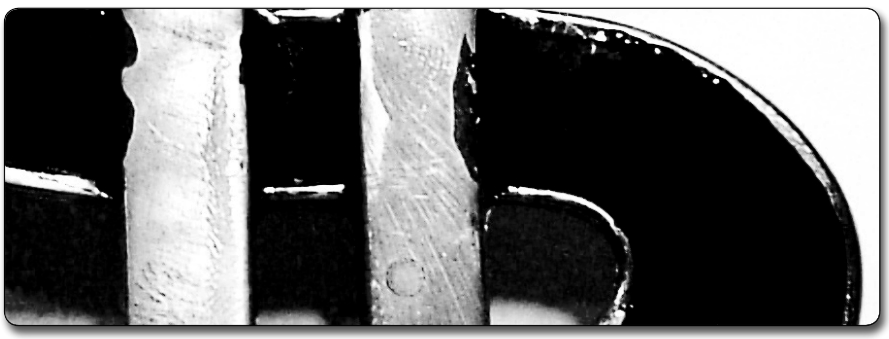


Figura 21. Vinculando os DataSources



Na configuração dos campos, temos a opção *Select a data field to display in the DropDownList*, onde configuramos a coluna responsável pelos dados que serão visíveis ao usuário (nesse caso *Descricao*) e o *Select a data field for the value of the DropDownList*, onde configuramos a coluna chave, que será *CategoriaID* (**Figura 20**).

Com o *DropDownList* preenchido, devemos vincular o controle ao *DataSource* de produtos, via *DataBinding*. Selecione novamente a *Smart Tasks* e escolha *Edit DataBinding*. Configure o evento *Selected Value* com o campo *CategoriaID* (**Figura 21**).

> **Nota**: Isso cria algo semelhante a um DBLookupComboBox, para quem está acostumado com a VCL.

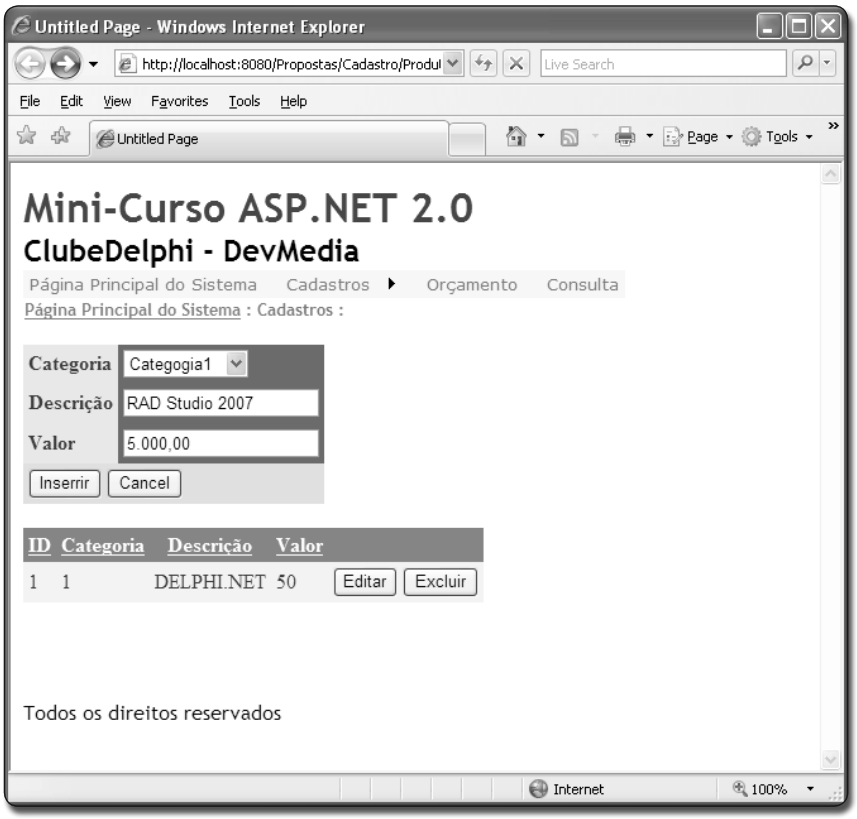


Figura 22. Finalização dos cadastros

Clique com o botão direito do mouse sobre o *DetailsView* e escolha *End Template Editing*. Neste momento finalizamos o formulário para inserção de dados. Seguindo o padrão do cadastro de categorias, devemos adicionar o *GridView* para exibir os valores da tabela, adicionando-o abaixo do *DetailsView*. Como já criamos uma *SqlDataSource* da tabela de produtos, podemos compartilhar a utilização do *SqlDataSource*.

Na *Smart Tasks* do *GridView*, selecione em *Choose Data Source* o *SqlDataSource* já criado ("SqlProduto"). Selecione as opções de paginação e ordenação conforme já realizamos no cadastro de categorias. No *Project Manager*, clique com o botão direito do mouse sobre o formulário de produtos e escolha *View in Browser* (**Figura 22**).

Conclusão

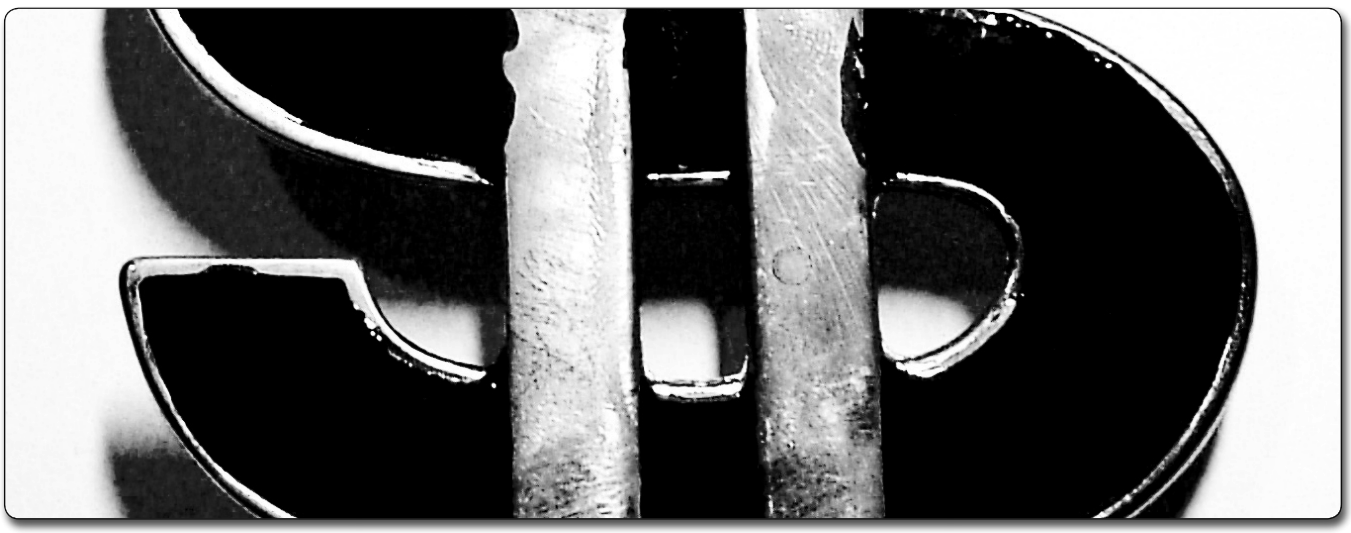
Os novos controles do ASP.NET 2.0 facilitam o desenvolvimento de aplicações Web juntamente com o RAD Studio 2007. Não perca na próxima edição a finalização da aplicação com muitos outros recursos do ASP.NET 2.0. Fique ligado! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



POR QUE A ALOG É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Desenvolvendo uma Aplicação Completa com PocketStudio

Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 4

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atual como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

Continuando o artigo da edição anterior, iremos criar a última tela de nossa aplicação que será a tela de digitação de itens do pedido. Após concluirmos a criação da mesma iremos desenvolver um conduit para sincronizarmos com o *Desktop* e trocarmos informações. O objetivo do nosso conduit será enviar para o Desktop as tabelas de *itens de pedido* e *pedidos*, e receber do *Desktop* as tabelas de *produtos* e *clientes*. Sempre que o usuário conectar o cabo USB no Palm e no Desktop selecionar o *HotSync* o sincronismo será realizado e os dados atualizados, a regra é sempre primeiro enviarmos os dados do Palm para o Desktop e depois recebermos os dados do Desktop para o Palm. Mãos à obra, vamos criar a tela de *Itens de Pedido* para depois entrarmos em detalhes do Conduit.

Criando a tela de digitação de itens do pedido

Vamos adicionar um novo formulário para criarmos a tela de digitação de itens do pedido, no menu principal do PocketStudio em *File|New>Form*, após adicionarmos o mesmo vamos salvá-lo clicando em *File|Save* ou pelas teclas de atalho *CTRL + S*, salvaremos ele com o nome "UfrmPedidoItem.pas" para indicarmos que é a *Unit* de itens do pedido. Feito isso altere seu *Name* para "FrmItensPedido" e *Caption* para "Itens do Pedido".

Desenharemos uma tela semelhante à **Figura 1**. Para isso inclua (todos da paleta *Forms*) um componente *List*("LstItens"), quatro *Label's* com os *Caption's* "Total do Pedido", "Qtde." e "Preço". Altere a propriedade fonte do primeiro Label para *ftBold*. Em seguida adicione ao lado de cada *Label* um componente *Field* representando um campo da tabela. Incluiremos *FldTotPed, FldCodProd, FldDescricao, FldQtde* e *FldPreco*. Os botões chameos de *BtnVoltar, BtnProcurar, BtnNovo, BtnGravar* e *BtnExcluir*. Se optar por utilizar Bitmap's para representar as

ações nos botões, não esqueça de incluir quatro Bitmap's e nomeá-los como *BmpProcurar, BmpNovo, BmpGravar* e *BmpExcluir*. Caso tenha dúvidas da utilização de imagens nos botões, retorne aos artigos anteriores e veja como realizamos a tarefa de inclusão de Bitmap's nas telas anteriores.

Antes de codificarmos os botões vamos escrever algumas funções que irão nos auxiliar nas validações dos dados informados e limpar a tela quando o usuário clicar no botão *Novo* para incluir um novo item no pedido.

Além disso trabalharemos com um novo conceito, o de manipulação de listas. Logicamente o componente *List* que incluimos na janela listará os *Itens do Pedido* conforme adicionarmos. Existem várias maneiras de "alimentarmos" a lista para apresentar os itens. Uma delas seria fazer um laço("loop") e percorrer a tabela de itens do pedido adicionando-os à lista, para isso usaríamos a função *AddCopy* da *PSList*, mas isso é uma operação demorada, e que consome muita memória, sendo assim utilizaremos duas funções que são as chamadas "receitas de bolo", pois montamos a estrutura e podemos utilizar em todas as nossas listas alterando apenas as tabelas e os *Field's*.

A grande vantagem dessas funções é que elas não usam a memória, pois trabalham com os métodos de desenho na

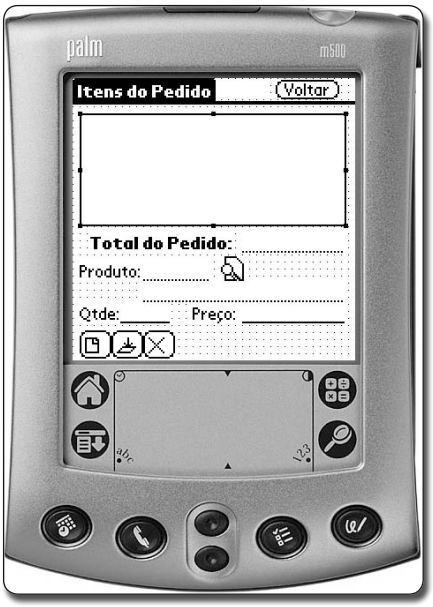


Figura 1. Tela dos itens do pedido

Listagem 1. Procedimentos para listagem, carregamento e limpeza das listas

```
procedure FListaItens(ItemNum: Int16; Bounds: RectanglePtr; var ItemsText: PChar);
var
  Buffer: Array[0..100] of Char;
  ItemInicial: Int16;
begin
  { Se o item solicitado for menor que ZERO, não fazemos nada! }
  if ItemNum < 0 then
    exit;
  { Nós movemos para o índice informado, dentro do
    banco de Itens tomando como base nosso o índice
    absoluto do primeiro registro do Pedido + o
    Número do Item a ser apresentado }
  ItemInicial := 0;
  PSDatabase.MoveTo(DBItp, ItemInicial + ItemNum);

  { Procuramos pelo Produto no banco de Produtos }
  if not ProdutosDB.ProcuraCodigo(PSDatabase.
    FieldUInt16(DBItp, Itp_CodProd)) then
    StrCopy(Buffer, 'Produto não encontrado!')
  else
    StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro, Prod_DescProd));
  { Desenhamos a Descrição do Produto }
  WinDrawTruncChars(Buffer, StrLen(Buffer),
    Bounds.TopLeft.x, Bounds.TopLeft.y,
    Bounds.Extent.x);
end;

procedure CarregaArrayItem;
var
  Numero: UInt16;
  Buffer: Array[0..30] of Char;
  RecNo: Int32;
  VlTotal: Double;
  NumItens: Int16;
begin
  { Zera Valor Total do Pedido }
  VlTotal := 0;
  { Zera variável de contagem de Itens }
  NumItens := 0;
  { Recupera o Código do Pedido }
  PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed)+ 1);
  { Utiliza a Busca Binária para encontrar o Primeiro Item }
  Numero := StrAToI(Buffer);
  RecNo := DBSeek(DBItp, Itp_NumeroPed, @Numero, ftUInt32, False);

  { Se não encontrar, não apresenta nada }
  if RecNo = -1 then
  begin
    { Valor Total do Pedido é ZERO }
    PSField.SetText(FldTotPed, '0.00');
    { Atualiza a Lista }
    LstSetListChoices(PSList.Ptr(LstItensPed), nil, NumItens);
    LstSetDrawFunction(PSList.Ptr(LstItensPed), @FListaItens);
    LstDrawList(PSList.Ptr(LstItensPed));
    exit;
  end;
  { Movemos o cursor do até o primeiro registro encontrado }
  PSDatabase.MoveTo(DBItp, RecNo);
  { Salva o índice absoluto do primeiro Item do Pedido }
  ItemInicial := RecNo;
  { Interage pelos registros do Banco e Carrega Array }
  while not PSDatabase.EOF(DBItp) do
  begin
    if PSDatabase.FieldUInt16(DBItp,Itp_NumeroPed) <> Numero then
      break;
    { Soma Valor Total }
    VlTotal := VlTotal +
      (PSDatabase.FieldDouble(DBItp, Itp_Preco) * PSDatabase.FieldUInt16(DBItp, Itp_Qtde));
    { Incrementamos o índice da Lista }
    Inc(NumItens);
    PSDatabase.Next(DBItp);
  end;
  { Coloca o Valor Total do Pedido no campo específico }?
  FormatFloat(Buffer, VlTotal, 2);
  PSField.SetText(FldTotPed, Buffer);
  { Atualiza a Lista }
  LstSetListChoices(PSList.Ptr(LstItensPed), nil, NumItens);
  LstSetDrawFunction(PSList.Ptr(LstItensPed), @FListaItens);
  LstDrawList(PSList.Ptr(LstItensPed));
end;

procedure Limpa;
begin
  { Limpamos os Campos do formulário }
  PSField.SetText(FldCodProd, '');
  PSField.SetText(FldDescricao, '');
  PSField.SetText(FldQtde, '');
  PSField.SetText(FldPreco, '');
end;
```

Listagem 2. Código de validação

```
function Valida: Boolean;
var
  Buffer: Array[0..30] of Char;
begin
  Result := False;
  { Validamos se o campo Código do Produto é válido }
  PSField.Text(FldCodProd,Buffer,PSField.TextLength(FldCodProd)+1);

  if not ProdutosDB.ProcuraCodigo(StrAToI(Buffer)) then
  begin
    ShowMessage('Produto Inválido!');
    Exit;
  end;

  { Verificamos se a quantidade é maior do que Zero }
  PSField.Text(FldQtde, Buffer, PSField.TextLength(FldQtde) + 1);
  if StrAToI(Buffer) <= 0 then
  begin
    ShowMessage('Quantidade Inválida!');
    PSField.SetFocused(FldQtde, True);
    Exit;
  end;

  { Verificamos se o valor é maior que zero }
  PSField.Text(FldPreco,Buffer,PSField.TextLength(FldPreco) + 1);
  if FLPAToF(Buffer)<=0 then
  begin
    ShowMessage('Valor Unitário Inválido!');
    PSField.SetFocused(FldPreco, True);
    Exit;
  end;
  Result := True;
end;
```

Listagem 3. Código da procedure MostraRegistro

```
procedure MostraRegistro;
var
  Buffer, StrQtdEmb: Array[0..30] of Char;
begin
  { Usamos as funções da PSDatabase(FieldUInt16, FieldStringPtr, etc.) para recuperar a
    informação dos campos }
  { Código do Produto }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBItp, Itp_CodProd));
  PSField.SetText(FldCodProd, Buffer);

  { Mostramos os dados do Produto no formulário }
  if ProdutosDB.ProcuraCodigo(StrAToI(Buffer)) then
  begin
    { Código }
    StrIToA(Buffer, PSDatabase.FieldUInt16(DBPro, Prod_Codigo));
    PSField.SetText(FldCodPRod, Buffer);

    { Descrição }
    PSField.SetText(FldDescricao, PSDatabase.
      FieldStringPtr(DBPro, Prod_DescProd));

    { Unidade }
    StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro, Prod_UN));
    PSField.SetText(FldPdiUnidade, Buffer);
  end;
  { Quantidade }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBItp, Itp_Qtde));
  PSField.SetText(FldQtde, Buffer);

  { Vamos formatar o valor do campo Itp_Preco antes de mostrá-lo
    O PalmOS não tem função de formatação de campos com casas decimais,
    por isso usamos uma função customizada chamada FormatFloat e colocamos o
    valor formatado no Field }
  FormatFloat(Buffer, PSDatabase.FieldDouble(DBItp, Itp_Preco), 2);
  PSField.SetText(FldPreco, Buffer);

  { Se mostramos dados de um registro, não estamos incluindo }
  { Sendo assim setamos a variável de controle do banco de dados }
  ItemPedDB.bItemPedInclui := False;
end;
```

tela como o *WinDrawTruncChars***.** Dessa forma os itens são desenhados na janela e não inseridos na lista o que é extremamente rápido, com a vantagem de usar pouquíssima memória. Por isso, inclua as procedures *FListaItens* e *CarregaArrayItem* à área *Implementation* do formulário de Itens de Pedido. Logo em seguida inclua o procedimento *Limpa*, todas presentes na **Listagem 1**. Fiz questão de comentar cada linha no próprio código, portanto veja as explicações das listagens em formato de comentários no código-fonte.

Para garantirmos a integridade dos dados vamos criar uma função para validação dos dados antes de salvarmos os mesmos. Faremos consistências simples, mas importantes, como por exemplo se o código do produtos existe, se a quantidade informada é maior que zero e se o preço informado é maior que zero. Essas validações garantem dados íntegros. O código dela está na **Listagem 2**, também devidamente comentada.

Outra função importante para nossa *Unit* é a *MostraRegistro*. Como seu nome diz ela irá mostrar os dados do produto. (**Listagem 3**).

Sempre que clicarmos no botão de consultar iremos chamar a função *ConsultaPro*, ela recebe como parâmetro um código do tipo *Uint16*, e utiliza a função *ProcuraCodigo* da base de dados para localizar o produto. Após isso mostra na tela as informações e coloca o foco no *Field* da quantidade (**Listagem 4**).

> **Nota**: Não esqueça de que para compilar o sistema, a *Unit* do formulário de Itens de Pedido precisa ter acesso às demais *Unit's* do projeto. Por isso, encontre a palavra *Implementation* e, abaixo dela, declare uma seção *Uses* se não existir e nela inclua as *Unit's*: *PedidosDb, ProdutosDb, LibAll, uCabPedido, uDadosProdutos* e *ItemPedDb*.

Após criarmos todas essas funções para auxiliarmos no perfeito funcionamento da tela de *Itens do Pedido* vamos codificar os botões, iniciando pelo botão de Voltar. Sua codificação é simples e já a fizemos em outras telas. Basta fazer uma chamada ao método *FrmGotoForm* informando qual o formulário que deverá

ser chamado, nesse caso o formulário principal. Veja a seguir:

```
FrmGotoForm(FrmCabPedido);
```

Já o *BtnConsultaPro*, fará a consulta no banco para encontrar um produto. Utilizamos nesse caso uma variável Buffer para armazenar o código do produto procurado e informar a procedure *ConsultaPro* para pesquisa. Digite o código da **Listagem 5** no evento *OnClick* do botão em questão.

No código do botão *Novo* precisamos apenas configurar a variável de controle de banco de dados de *Itens* para indicarmos que estamos incluindo um novo registro e limpar a tela, para que o usuário possa entrar com as informações. Para que isso seja feito, basta incluir o código a seguir no botão *Novo*. Veja que o procedimento é bem simples.

```
ItemPedDB.ItemPedInclui:= True;
Limpa;
```

Sem dúvida o código mais trabalhoso será o código do botão *Gravar*, onde passaremos o valor dos *Field's* para os campos da base de dados *ItemPedDB*. Antes disso note que estamos utilizando a função de validação. Confira na **Listagem 6**

Listagem 4. Código da função de consulta

```
procedure ConsultaPro(Codigo: UInt16);
var
  Buffer, StrQtdEmb: Array[0..40] of Char;
begin
  { Procura pelo Produto }
  if not ProdutosDB.ProcuraCodigo(Codigo) then
  begin
    ShowMessage('Produto não encontrado!');
    exit;
  end;
  { Converte o Código do Produto }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBPro,
    Prod_Codigo));
  PSField.SetText(FldCodProd, Buffer);
  { Descrição }
  PSField.SetText(FldDescricao, PSDatabase.
    FieldStringPtr(DBPro, Prod_DescProd));
  { Unidade }
  StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro,
    Prod_Un));
  PSField.SetText(FldPdiUnidade, Buffer);
  { Preço }
  FormatFloat(Buffer, PSDatabase.FieldDouble(DBPro,
    Prod_Preco), 2);
  PSField.SetText(FldPreco, Buffer);
  { Coloca o foco no campo Quantidade }
  PSField.SetFocused(FldQtde, True);
end;
```

Listagem 5. Código do botão BtnConsultaPro

```
procedure BtnConsultaProSelect;
var
 Buffer: Array[0..10] of Char;
 Codigo: UInt16;
begin
  { Recuperamos o valor do FieldCodProd e passamos
    para a variável Buffer }
  PsField.Text(FldCodProd, Buffer, PsField.
    TextLength(FldCodProd) + 1);
  { Convertemos o valor da variável Buffer de
    String para Integer }
  Codigo:= StrAToI(Buffer);
  { Por fim chamamos a função ConsultaPro passando
    a variável Codigo como parâmetro }
  ConsultaPro(Codigo);
end;
```

Listagem 6. Código do botão gravar

```
procedure BtnGravar;
var
  Buffer: Array[0..30] of Char;
  Codigo, RecPos: UInt16;
begin

  { Valida os campos do formulário }
  if not Valida then
    exit;

  { Se estivermos Incluindo colocamos o registro em modo de edição }
  if ItemPedDB.bItemPedInclui then
  begin

    { Recuperamos o Número do Pedido }
    PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed) + 1);
    Codigo := StrAToI(Buffer);

    { Recuperamos a posição que o registro deve ser inserido antes de adicionar o novo Registro }
    RecPos := DBInsertionPos(DBItp, Itp_NumeroPed, @Codigo, ftUInt32, True);
    PSDatabase.Append(DBItp);
  end
  else
    PSDatabase.Edit(DBItp);

  { Pegamos os dados dos campos do Form e colocamos nos campos do }
  { banco de dados de acordo com o Índice deles }
  PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed) + 1);
  PSDatabase.SetFieldUInt32(DBItp, Itp_NumeroPed, StrAToI(Buffer));

  { Código do Produto }
  PSField.Text(FldCodProd, Buffer, PSField.TextLength(FldCodProd) + 1);
  PSDatabase.SetFieldUInt16(DBItp, Itp_CodProd, StrAToI(Buffer));

  { Quantidade }
  PSField.Text(FldQtde, Buffer, PSField.TextLength(FldQtde) + 1);
  PSDatabase.SetFieldUInt16(DBItp, Itp_Qtde, StrAToI(Buffer));

  { Valor Unitário }
  PSField.Text(FldPreco, Buffer, PSField. TextLength(FldPreco) + 1);
  PSDatabase.SetFieldDouble(DBItp ,Itp_Preco, FLPAToF(Buffer));

  { Grava as informações no banco de dados }
  PSDatabase.Post(DBItp);

  { Depois de Incluir, moveremos o registro para o local correto, }
  { encontrado anteriormente pela função DBInsertionPos }
  if ItemPedDB.bItemPedInclui then
    DBMoveRecord(DBItp, RecPos);

  { Voltamos a incluir um Registro }
  ItemPedDB.bItemPedInclui := True;

  { Limpamos os campos do Item }
  Limpa;

  { Carrega novamente o Array para refletir a mudança no banco de dados de Itens de Pedido }
  CarregaArrayItem;
end;
```

Listagem 7. Código do botão excluir

```
procedure BtPdiExcluirSelect;
begin
  { Se estivermos incluindo, retornamos um erro }
  if PedidosItemDB.bPedidoItemInclui then
  begin
    ShowMessage('Não existe Item selecionado!');
    exit;
  end;

  { Solicita confirmação }
  if FrmAlert(AlertConfirma)=0 then
    Exit;

  { Apaga o Registro }
  PSDatabase.Delete(DBPdi);

  { Voltamos a incluir um Registro }
  PedidosItemDB.bPedidoItemInclui := True;

  { Limpamos os campos do Item }
  Limpa;

  { Carrega novamente o Array para refletir a
    mudança no banco de dados }
  { de Itens de Pedido }
  CarregaArrayItem;
end;
```

o código completo do botão Gravar. O código está totalmente comentado para que possa entender perfeitamente tudo o que estamos fazendo.

O último botão a ser codificado é botão de Excluir. Basicamente verificamos o estado da tabela, que pode estar em inserção ou alteração, nesse caso retornamos uma mensagem de erro ao usuário e abortamos. Caso contrário solicitamos uma confirmação do usuário e preparamos para exclusão. O segredo todo está na chamada ao método *DBMoveRecord* que faz efetivamente a exclusão no banco de dados. Em seguida apenas "informamos" ao banco que deve retornar ao estado normal. Por fim recarregamos os dados em tela (**Listagem 7**).

Para finalizarmos a codificação da *Unit* de *Itens do Pedido*, vamos codificar o evento *OnOpen* do *Form*. Selecionando o formulário, vamos ao *Object Inspector* na aba *Events*. Clique duas vezes no evento *OnOpen* e vamos codificá-lo conforme o código da **Listagem 8**.

Construindo o conduit

O primeiro passo é instalarmos o *TurboSync* no Delphi. Para isso abrimos o Delphi 7 e selecionamos *Component|Install Packages*. Em seguida clique em *Add* e localize o arquivo *TurboSyncD7Trial.bpl* no diretório onde descompactou os arquivos baixados e descompactados anteriormente.

Após a instalação do pacote, vamos no menu *File|New>Other>New>Dll Wizard*. O próximo passo é adicionarmos um *Data Module* que será o objeto responsável por armazenar nossos componentes de acesso a dados. Use o menu *File|New>Data Module* e salve-o como "uDmConduit.pas". Criada a dll, devemos adicionar a *Unit Forms* na *Uses* da dll, e na seção *begin..end* criarmos nosso *Data Module* quando executada. Salve nosso projeto como "ConduitDelphi.dpr".

Feito isso devemos adicionar os componentes que farão a transferência dos dados. Após a instalação do *TurboSync*, foi criada uma aba chamada *Conduit*, localize-a e adicione o primeiro componente que é o *TabdConduit*, e o *TabdConduitDB*. O conduit deve ter apenas um *TabdConduit*, e outros

TabdConduitDB para cada tabela que será sincronizada. Para o nosso exemplo devemos adicionar os seguintes componentes configurados conforme a **Figura 2**.

Note que estamos trabalhando com os componentes da paleta BDE do Delphi. Podemos utilizar qualquer base de dados para fazer a importação para o banco de dados do *Desktop*. Na **Figura 2** temos a imagem do *DMConduit* com os componentes adicionados.

Os componentes *Table* da paleta BDE estarão conectados através de um *alias* que irá apontar para uma base de dados Paradox. Não vou entrar em detalhes aqui sobre a criação do *alias* e das tabelas do *Desktop*. A próxima configuração que devemos fazer é criar os *Field's* dos componentes *TabdConduitDB*, *ItemPedDB* e *PedidosDB*, é uma etapa muito importante pois precisamos analisar o tipo de cada campo no PalmOS para definir o mesmo no componente. Os campos são criados na propriedade *Field's* do componente, clicando em *Add*. O primeiro que vamos montar é o do *PedidosDB*, devemos configurar os campos dos *Field's Editor's* de *PedidosDb* e *ItensPedidoDB* como mostrado nas **Tabelas 1** e **2**. Para isso selecione o *TTabdConduitDb* apropriado, clique duas vezes em sua propriedade *Field's* e na janela que se abre adicione os campos necessários.

Em seguida faça as devidas configurações para criação das tabelas em Paradox e inclua nos *Field's Editor's* de cada *TTable*

Campo	Tipo
Itp_NumeroPed	pftLongint
Itp_CodProd	pftSmallint
Itp_Qtde	pftSmallint
Itp_Preco	pftDouble

Figura 2. Tipos dos Fields do ItemPedDB

Campo	Tipo
Ped_NumeroPedido	pftLongint
Ped_CodCli	pftSmallint
Ped_Emissao	pftDate
Ped_Cond	pftSmallint
Ped_Total	pftDouble

Tabela 1. Fields Editor da tabela PedidosDB

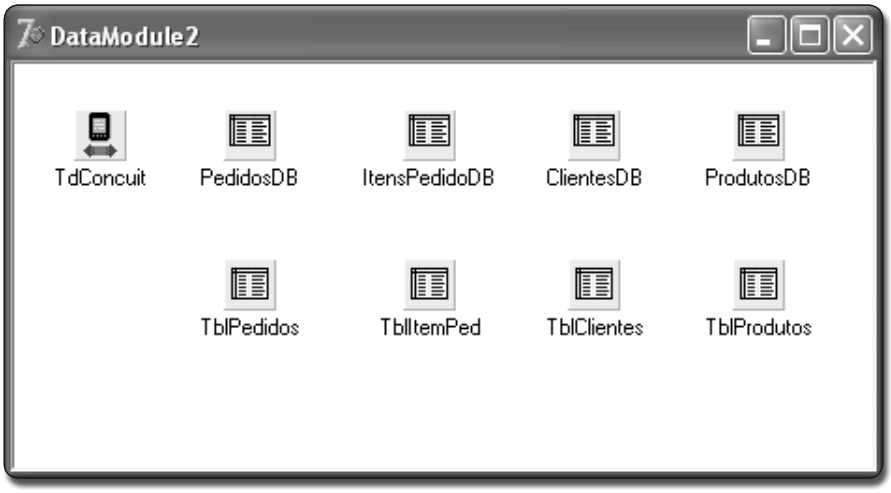


Figura 2. Data Module do nosso aplicativo conduit

Listagem 8. Código do evento OnOpen do formulário

```
procedure FrmItemsPedidoOpen;
var
  Buffer: Array[0..30] of Char;
begin
  PSForm.Draw;

  { Carrega número do Pedido no campo do formulário de Itens de Pedido }
  StrIToA(Buffer, PedidoAtual);
  PSField.SetText(FldNumPed, Buffer);

  { Carrega os Itens do Pedido }
  CarregaArrayItem;

  { Coloca o Banco de Dados em Modo de Inclusão }
  ItemPedDB.bItemPedInclui := True;
end;
```

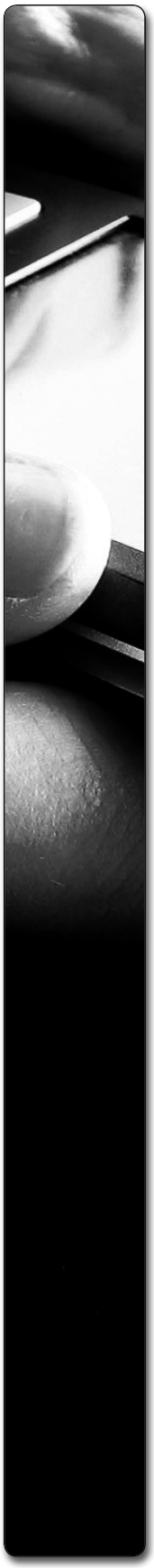
Listagem 9. Exemplo de importação de dados

```
procedure TDataModule2.DataModuleCreate(Sender: TObject);
begin
  PedidosDb.Open;
  TblPedidos.Open;
  while not PedidosDb.EOF do
  begin
    with TblPedidos, PedidosDB do
    begin
      TblPedidos.Append;
      FieldByName('Ped_NumeroPedido').AsFloat := FieldByName('Ped_NumeroPedido').AsFloat;
      FieldByName('Ped_CodCli').AsFloat := FieldByName('Ped_CodCli').AsFloat;
      FieldByName('Ped_Emissao').AsDateTime := FieldByName('Ped_Emissao').AsDateTime;
      FieldByName('Ped_Cond').AsFloat := FieldByName('Ped_Cond').AsFloat;
      FieldByName('Ped_Total').AsFloat := FieldByName('Ped_Total').AsFloat;
      Post;
      Next;
    end;
  end;
end;
```

Nota do DevMan

Para que o componente TurboSync seja instalado corretamente em seu Delphi, será necessário ter as dll's sync20.dll e hslog.dll. Esses arquivos são instalados juntamente com o aplicativo Palm Desktop. O Palm Desktop é um programa utilizado para sincronizarmos o dispositivo móvel com o Desktop. Com ele podemos inclusive cadastrar contatos, endereços, telefones, e quaisquer outros dados diretamente no computador e transferi-los para o dispositivo móvel, poupando tempo.

O Palm Desktop é encontrado no CD de instalação do seu Palm, caso não tenha um Palm, poderá baixar gratuitamente em sites ligados a comunidades que usufruem desse tipo de dispositivo, como por exemplo o www.palmbrasil.com.br que possui uma enorme quantidade de aplicativos e utilitários tanto para o Palm quanto para Desktop.



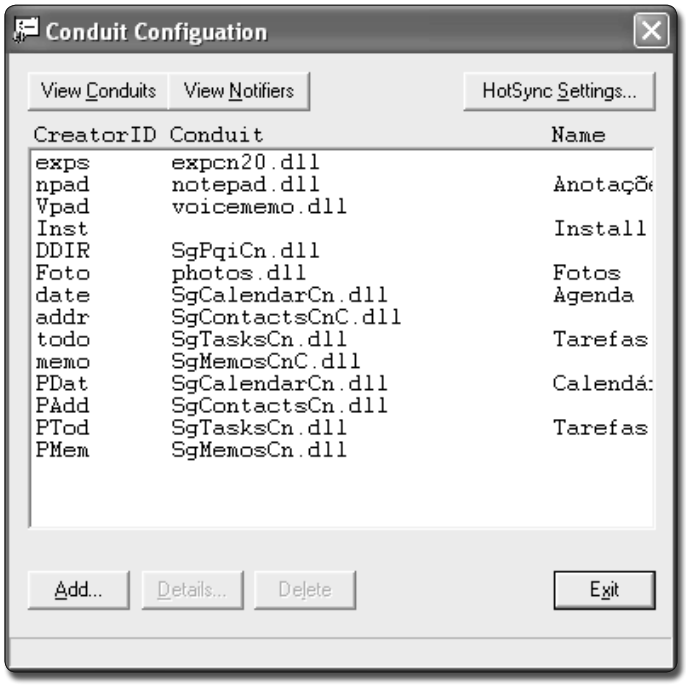


Figura 3. Tela de registro do conduit

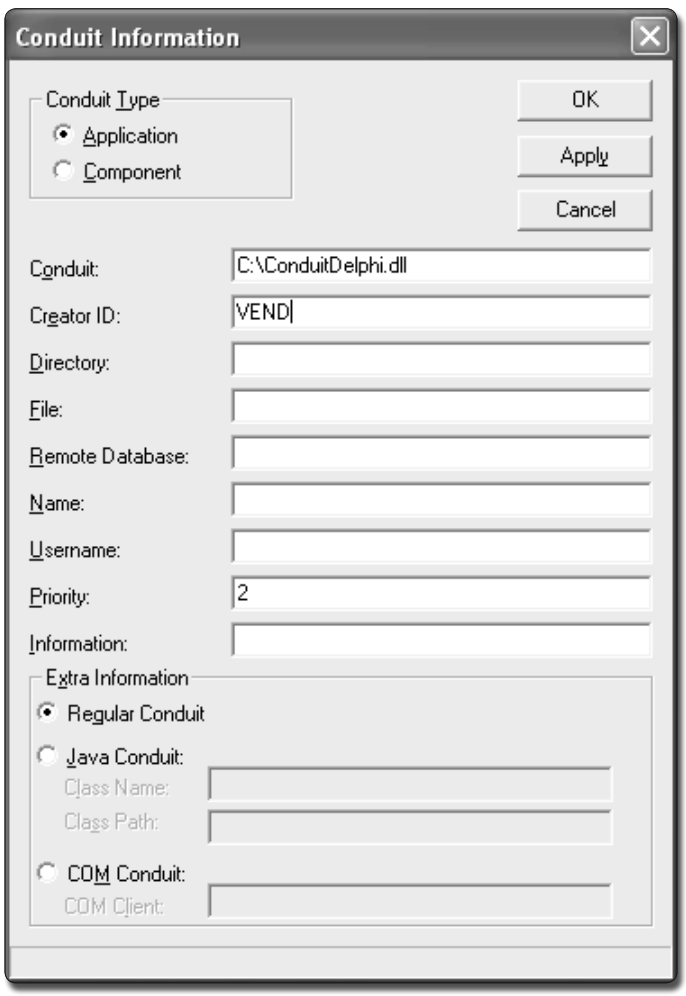


Figura 4. Tela de registro do conduit

o campos necessários. Feita a configuração dos campos dos componentes *TabdConduitDB*, e criadas as tabelas do Paradox no *Desktop*, devemos criar funções para ler os dados das tabelas do *Palm* e gravar nas tabelas no Desktop e vice-versa. Lembrando que a aplicação que construímos possui quatro tabelas, *Pedidos* e *Itens* que serão sincronizados do *Palm* para o *Desktop* e *Produtos* e *Clientes* que serão sincronizados do *Desktop* para o *Palm*.

Como estamos trabalhando com *DataSets*, lembrando que o componente *TabdConduitDB* é um *DataSet*, será um processo muito simples. Essas funções devem ser criadas na *Unit* do *DmConduit*, para cada tabela devemos criar uma função para facilitar o entendimento e a manutenção quando necessário. Os fontes do conduit estão disponíveis juntamente com os exemplos deste artigo. Não vou entrar em detalhes aqui sobre a construção das funções, pois os fontes estão comentados e fáceis de entender. Porém, para que se tenha uma noção do que é feito veja a importação de uma das tabelas do sistema na **Listagem 9**. Note que estamos usando apenas os métodos padrões de *TDataSet*, tais como *Insert* e *Edit*, em conjunto com um laço *while not...eof do*. O mais importante do *conduit* é entender o seu funcionamento.

Conduit

Após concluirmos o desenvolvimento de nossa aplicação para o Palm, partimos para o desenvolvimento da aplicação *Conduit*. Uma aplicação *conduit* pode ser desenvolvida em qualquer linguagem que tenha suporte a objetos COM e existem vários exemplos na internet. Uma aplicação dessas, como seu nome já diz, tem a função de exportar os dados do *Desktop* para o *Palm* e do *Palm* para o *Desktop*. Nosso exemplo será construído utilizando o Delphi 7, e um componente chamado *TurboSync*. Utilizando esse componente trabalharemos da mesma forma que trabalharmos com um *DataSet* do Delphi. O *TurboSync* pode ser baixado no endereço *www.tabdee.ltd.uk/Downloads/TurboSync.html*.

O conduit é muito simples de ser instalado por ser um pacote ("BPL"), a única consideração com relação ao componente disponibilizado, é que por

ser *Trial* ele funcionará apenas enquanto o Delphi estiver rodando, sendo assim devemos lembrar desse detalhe na hora que fazermos o sincronismo.

A aplicação que iremos desenvolver é uma dll. E para ficar claro como o *Palm* irá "saber" que precisa executá-la, o segredo é registrarmos a mesma no processo do *HotSync*.

Para registrarmos nossa dll no processo do *HotSync* devemos adicioná-la no registro do Windows. Para isso estou disponibilizando um executável, chamado de *RegRegistro.exe*, que irá fazer o trabalho para nós, já que alterar o registro do Windows é uma operação delicada. O aplicativo encontra-se disponível juntamente com os fontes de nossa aplicação no site da DevMedia.

Outro segredo para que o sincronismo funcione perfeitamente é que a dll registrada tenha o mesmo *CreatorID* registrado no registro do Windows e na aplicação instalada no *Palm* e informar a pasta onde ela está. A **Figura 3** mostra o nosso *conduit* já registrado, sendo *CreatorID* o *creator* da aplicação, que definimos na *Unit* principal do projeto, ou seja, *Vendas*. Para facilitar sua localização basta procurarmos pela diretiva {$CREATOR 'VEND'}. O *Conduit* é a pasta onde está a dll que iremos criar.

Para que possamos então fazer o registro da aplicação, basta abrir o arquivo *RegRegistro.exe* (**Figura 4**) e clicar em *Add*. Uma nova janela será aberta e é nesse momento que preenchemos os campos *Conduit, Creator ID* e *Priority*.

Conclusão

Neste artigo finalizamos nossa aplicação de vendas, construindo a tela para digitarmos os itens do pedido e o *Conduit*. Nos quatro artigos sobre desenvolvimento de aplicações com PocketStudio, desenvolvemos uma aplicação simples para controle de vendas, porém abordando os principais conceitos do desenvolvimento para PalmOS com PocketStudio. A intenção dos artigos não foi desenvolver uma aplicação robusta e comercial e sim mostrar o caminho para tal. Com a aplicação construída com certeza você terá condições de desenvolver a sua aplicação robusta e comercial aplicando os conceitos aqui apresentados. Grande abraço a todos e até a próxima. ●

Nota do DevMan

HotSync é um conduit pré-instalado juntamente com o aplicativo PalmDesktop, que vem com o CD de instalação do dispositivo. Sempre que executarmos o sincronismo entre o Palm e o Desktop o HotSync será responsável por controlar o que será sincronizado.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Controle on-line de vídeo-locadora - Parte 3

Veja como criar um sistema on-line de controle para uma vídeo-locadora

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Continuando o desenvolvimento da nossa aplicação ASP.NET com Delphi for .NET, sobre um sistema on-line de vídeo-locadora, veremos agora como confirmar a locação a partir de uma reserva e fazer a locação de vídeos. Mão na massa!

Confirmando a locação a partir de uma reserva

Realizada a reserva através da página do usuário, faremos agora uma página destinada à área administrativa do sistema para concretizar uma reserva transformando-a para uma locação, supondo que ocorra a retirada do vídeo. Caso a retirada do vídeo não ocorra no prazo previsto, a mesma será ignorada nas consultas a seguir levando em consideração a data prevista e a data atual. No menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* crie uma nova página com o nome de "a_reservas.aspx" e adicione ao corpo dela uma tabela de 4 linhas e 1 coluna com 700 pixels de largura. Arraste para a primeira linha da tabela o *uccontrole.ascx* que fará o controle de usuários autenticados nesta página. Na segunda linha escreva o texto de identificação "Manutenção de Reservas" e na linha seguinte adicione um *DataGrid*("gridReservas").

Adicione por final um *Button* ("btn Voltar"). Selecionando o *gridReservas*, clique sobre o mesmo com o botão direito do *mouse* e selecione o item *Auto Format* aplicando a ele um estilo de cores e fontes. Clicando novamente selecione agora o item *Property Builder* onde iremos modificar a estrutura da categoria *Columns*, onde no topo da janela desmarque o item *Create columns automatically at run time*. Em seguida clique no item *Bound Column* em *Available columns* e envie-o para *Selected columns*. Agora selecione-o e atribua o valor *COD_LOCACAO* na propriedade *Data Field*, onde receberemos o valor do campo na tabela *Locacao*

através do *result set* da *Select*. Desmarque também a opção *Visible* da coluna.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* ("Texto de cabeçalho"), digite "Titulo" e em *Data Field* o valor TITULO também resultante do *Select*. Arraste uma nova coluna do *Bound Column* para *Selected columns*, na propriedade *Header text* digite "Cliente" e em *Data Field* o valor "NOME", resultado do *Select*. Adicione ainda uma nova coluna e altere sua propriedade *Header text* para "Data Reserva". em *Data Field* digite o valor "DATA" e em *Data Formatting Expression* o valor "{0:dd/MM/yyyy}". Por fim inclua uma nova coluna de botões, chamada de *Button Column*. Expanda o *Button Column* e selecione a opção *Select*. Envie-o para a direita e modifique a propriedade *Header text* e para "Locação" e o *Text* para "Confirmar" e clique em *Ok* para confirmar nossas alterações.

Utilizando a tecla de atalho *F12*, vá até o código da página e declare na seção *Private* do código a *procedure* "CriaListaReservas" que irá preencher o *gridReservas* com os registros válidos encontrados no *Select* ao banco de dados. Utilize a teclas *Ctrl+Shift+C* para criar a implementação do procedimento e adicione ao mesmo o código da **Listagem 1.** O método criado é muito semelhante aos outros já criados, utiliza de componentes criados e instanciados em *runtime* no objeto *SelectCommand* do *DataAdapter* e passamos a instrução de *SQL* que obedecerá os parâmetros de registros com *Status* igual a "R" com a data prevista de devolução/vencimento maior ou igual a data atual ("*Today*");

Para que as informações sejam carregadas logo no primeiro momento de *Load* da página, adicione ao evento *Load* o código a seguir que irá realizar a chamada ao método responsável pela pesquisa e preenchimento das informações:

```
CriaListaReservas();
```

A confirmação de uma reserva, transformando-a em locação irá apenas ocorrer ao clicar sobre o *link* "Confirmar" da linha de reserva desejada. A confirmação envolve o evento *ItemCommand* do *gridReservas*, onde deverá ser adicionado o código da **Listagem 2**, criando os componentes em *runtime* e atribuindo um bloco

Listagem 1. Método para carregamento das reservas ativas

```
procedure TWebForm2.CriaListaReservas;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;

  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;

  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT LOCACAO.COD_LOCACAO, VIDEOS.TITULO,
    'CLIENTES.NOME, ' +
    'LOCACAO.DATA FROM LOCACAO ' +
    '   INNER JOIN VIDEOS ON (LOCACAO.VIDEO = 'VIDEOS.COD_VIDEO) ' +
    '   INNER JOIN CLIENTES ON (LOCACAO.CLIENTE = 'CLIENTES.COD_CLIENTE) ' +
    'WHERE ((LOCACAO.STATUS = ''R'') AND (LOCACAO
    '.PREV_DEVOLUCAO >= ''TODAY'')) ORDER BY
    'LOCACAO.DATA';

  { Criação em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');
  try
    { Carrega o resultado para o grid }
    gridReservas.DataSource := Ds;
    gridReservas.DataBind;
  finally
    Conn.Close;
  end;
end;
```

Listagem 2. Confirmação de Reserva

```
procedure TWebForm2.gridReservas_ItemCommand(
  source: System.Object; e: System.Web.UI.WebControls.DataGridCommandEventArgs);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
  prStatus : FbParameter;
  prDev    : FbParameter;
  prId_Loc : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;

  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;

  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText := 'UPDATE LOCACAO SET STATUS = ?, PREV_DEVOLUCAO  ' +
    ' = ? WHERE LOCACAO.COD_LOCACAO = ?';
  prStatus := FbParameter.Create;
  prDev    := FbParameter.Create;
  prId_Loc := FbParameter.Create;
  DataAdapter.UpdateCommand.Parameters.Add(prStatus);
  DataAdapter.UpdateCommand.Parameters.Add(prDev);
  DataAdapter.UpdateCommand.Parameters.Add(prId_Loc);
  DataAdapter.UpdateCommand.Parameters[0].Value := 'N';
  DataAdapter.UpdateCommand.Parameters[1].Value := System.&String.Format('{0:dd/MM/yyyy}',
    IncDay(DateTime.Today, 7));{Uses DateUtils}
  DataAdapter.UpdateCommand.Parameters[2].Value := e.Item.Cells[0].Text;
  if DataAdapter.UpdateCommand.ExecuteNonQuery> 0 then
  begin
    CriaListaReservas
  end;
end;
```

SQL ao objeto *UpdateCommand* do *Data Adapter*, também a criação de configuração dos parâmetros necessários. Ao final, no sucesso da operação será chamado novamente o método *CriaListaReservas* fazendo com que a página seja recarregada com informações atualizadas.

Por fim, adicione ainda ao evento *Click* do *btnVoltar* o código a seguir:

```
Response.Redirect('principal.aspx');
```

Que irá redirecionar o administrador de volta à página principal e na página *principal.aspx*, adicione ao evento *OnClick* do *btnReservas* o código

```
Response.Redirect('a_reservas.aspx');
```

Criando um *link* para a manutenção das reservas. Execute o sistema e confirme as reservas que criamos há pouco através do *login* de usuários.

Locação de Vídeos

Sem dúvidas uma das páginas indispensáveis ao sistema é a de locação de vídeos. A área de locações de vídeos será criada de forma bastante simples a fim de que seja de fácil aprendizagem os códigos a serem utilizados. No menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* crie uma nova página chamada "a_locacao.aspx" e adicione ao seu corpo uma tabela de 4 linhas e 1 coluna com 700 pixels de largura. Arraste para a primeira da tabela o *uccontrole.ascx* que fará o controle de usuários autenticados nesta página. Na segunda linha escreva o texto de identificação "Manutenção de Locações" e na linha seguinte adicione um *DataGrid*("gridLocacao").

Na seguinte linha adicione uma nova tabela com 6 linhas e 2 colunas utilizando 100% da largura limitada pela linha da tabela anterior. Adicione na linha 1, coluna 2 o texto "Cadastro de Locações" e nas demais linhas, adicione textos e objetos de acordo com os itens da **Tabela 1**.

Linha/Coluna	Texto/Objeto
Linha 2/Coluna 1	Texto "Vídeo:"
Linha 2/Coluna 2	*DropDownList ("ddlVideos")*
Linha 3/Coluna 1	Texto "Cliente:"
Linha 3/Coluna 2	*DropDownList ("ddlClientes")*
Linha 4/Coluna 1	Texto "Data:"
Linha 4/Coluna 2	*TextBox ("txtData")*
Linha 5/Coluna 1	Texto "Previsão de Devolução:"
Linha 5/Coluna 2	*TextBox ("TxtPrevDev")*
Linha 6/Coluna 2	Button ("btnSalvar" e "btnVoltar")

Tabela 1. Lista de componentes

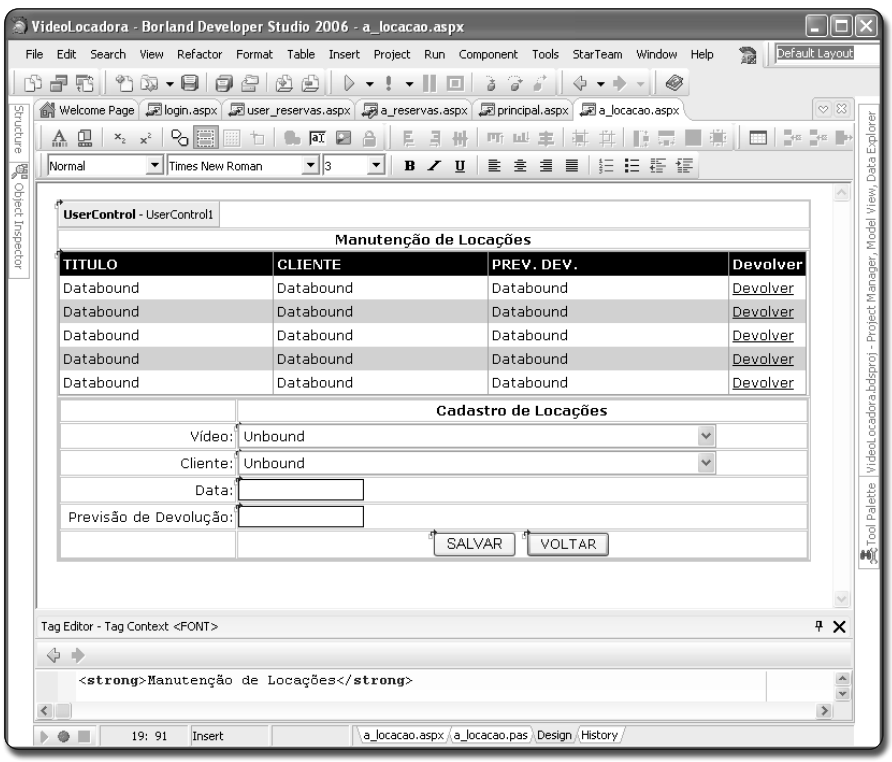


Figura 1. Layout da página de manutenção de Locações

Finalizada a adição de textos e componentes ao corpo da página, a mesma deverá ficar com um *layout* semelhante à **Figura 1**, onde passamos a ver agora os processos de codificação necessárias para a manutenção das locações, com devolução e cadastros de novos empréstimos.

Selecionando o *gridLocacao*, utilize o botão direito do mouse para aplicar um estilo de formatação através do item *Auto Format* e após isso selecione o *Property Builder* onde iremos modificar a estrutura da categoria *Columns*, onde no topo da janela desmarque o item *Create columns automatically at run time*. Em seguida clique em *Bound Column* em *Available columns* e envie-o para *Selected columns*. Agora selecione-o e atribua o valor *COD_LOCACAO* na propriedade *Data Field*, onde receberemos o valor do campo na tabela *Locacao* através do *result set* da *Select*. Desmarque também a opção *Visible* da coluna.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* ("Texto de cabeçalho"), digite "Titulo" e em *Data Field* o valor *TITULO* também resultante do *Select*. Arraste uma nova coluna do *Bound Column* para *Selected columns* e na propriedade *Header text* digite "Cliente" e em *Data Field* o valor "NOME", resultado do *Select*. Adicione ainda uma nova

coluna e altere sua propriedade *Header text* para "Prev. Dev". em *Data Field* digite o valor "PREV_DEVOLUCAO" e em *Data Formatting Expression* o valor "{0:dd/MM/yyyy}". Por fim inclua uma nova coluna de botões, chamada de *Button Column*. Expanda o *Button Column* e selecione a opção *Select*. Envie-o para a direita e modifique a propriedade *Header text* e *Text* para "Devolver" e clique em *Ok* para confirmar nossas alterações.

Temos na página de manutenção de locações 3 objetos que dependem de uma estrutura de *DataSet* para que suas informações sejam carregadas (*gridLocacoes*, *ddlVideos* e *ddlClientes*). Para cada um destes objetos necessitamos criar um procedimento, que através de uma pesquisa de SQL configurada ao objeto *SelectCommand* de um *DataAdapter* irá resultar em um bloco de dados, passados individualmente a cada um deles por meio do método *Fill* disponível no *DataAdapter*. Acessando o código da página, declare na seção *Private* três *procedures* conforme código a seguir e utilizando as teclas de atalho *Ctrl+Shift+C* crie a implementação dos métodos e adicione a cada um deles o código da **Listagem 3** de acordo com o nome de cada procedimento listado. Os métodos também utilizam componentes criados em *runtime* e os códigos encontram-se comentados conforme as funcionalidades dos blocos utilizados.

```
procedure CarregaLocacoes;
procedure CarregaVideos;
procedure CarregaClientes;
```

Se observar com atenção verá que utilizamos os mesmos mecanismos de seleção dos dados em todos os procedimentos. Criamos todos os componentes em tempo de execução, fazemos as *selects* necessárias e preenchemos os controles de tela.

Para que as informações sejam carregadas no primeiro momento da visualização da página, adicione ao evento *Load* o código da **Listagem 4**, onde verificamos se a página está sendo carregada ou apenas atualizada por meio do *IsPostBack* e chamamos então os procedimentos criados. Atribuímos ainda aos campos de texto os valores de data atual para a data da locação e data atual acrescidos de 7 dias para a data prevista de devolução. Neste caso você

Listagem 3. Procedimentos para carregar Locações, Vídeos e Clientes

```
procedure TWebForm1.CarregaLocacoes;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT LOCACAO.COD_LOCACAO, VIDEOS.TITULO, CLIENTES.NOME, LOCACAO.PREV_DEVOLUCAO FROM LOCACAO ' +
    '   INNER JOIN VIDEOS ON (LOCACAO.VIDEO = VIDEOS.COD_VIDEO) ' +
    '   INNER JOIN CLIENTES ON (LOCACAO.CLIENTE = CLIENTES.COD_CLIENTE) ' +
    'WHERE ((LOCACAO.STATUS = ''N'')) ORDER BY CLIENTES.NOME';
  { Criaçã em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');
  try
    gridLocacao.DataSource := Ds;
    gridLocacao.DataBind;
  finally
    Conn.Close;
  end;
end;
procedure TWebForm1.CarregaVideos;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT V.COD_VIDEO, V.TITULO FROM VIDEOS V '+
    'WHERE V.COD_VIDEO NOT IN (SELECT LOCACAO.VIDEO FROM LOCACAO '+
    'WHERE ((LOCACAO.VIDEO = V.COD_VIDEO) and ((LOCACAO.STATUS = ''R'') '+
    'OR (LOCACAO.STATUS= ''N'')))) ORDER BY V.TITULO';
  { Criaçã em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');
  try
    ddlVideos.DataSource := Ds;
    ddlVideos.DataBind;
  finally
    Conn.Close;
  end;
end;
procedure TWebForm1.CarregaClientes;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT CLIENTES.COD_CLIENTE, CLIENTES.NOME FROM CLIENTES ORDER BY CLIENTES.NOME';
  { Criaçã em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');
  try
    ddlClientes.DataSource := Ds;
    ddlClientes.DataBind;
  finally
    Conn.Close;
  end;
end;
```

poderá fazer uma verificação através da função *DayOfWeek* também disponível na *namespace DateUtils* a ser declarada e verificar se o dia previsto é diferente de sábado ou domingo.

Ao executar a página, será listada toda a reserva de vídeos ativa, onde você poderá implementar depois, mais as opções de filtros por data, por cliente, entre outros. Serão também carregados os vídeos disponíveis para locação bem como a lista de todos os clientes. Ambos os componentes possíveis de implementação de métodos de pesquisa que ficam por sua conta. A locação de novos vídeos está baseada nos itens selecionados na lista de vídeos e clientes levando também em consideração as datas de locação e previsão de devolução. O código responsável por adicionar uma nova locação deverá ser adicionado ao evento *OnClick* do *btnSalvar* e está listado e comentado na **Listagem 5**, utilizando-se de componentes criados em tempo de execução e uma instrução de *Insert* adicionada ao objeto *InsertCommand* do *DataAdapter* e da criação, configuração e atribuição de valores ao parâmetros necessários. Novamente recorra aos comentários distribuídos em todo o código-fonte para melhor entendimento.

Listagem 4. Código do evento Load da página

```
if not page.IsPostBack then
begin
  CarregaLocacoes;
  CarregaVideos;
  CarregaClientes;
  txtData.Text := DateTime.Today.ToString('dd/MM/yyyy');
  TxtPrevDev.Text := System.&String.Format(
    '{0:dd/MM/yyyy}', IncDay(DateTime.Today, 7));{Uses DateUtils}
end;
```

Listagem 5. Inclusão de nova locação

```
procedure TWebForm1.btnSalvar_Click(sender: System.Object; e: System.EventArgs);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
  prVideo : FbParameter;
  prCliente : FbParameter;
  prData : FbParameter;
  prStatus : FbParameter;
  prDev : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'INSERT INTO LOCACAO (VIDEO, CLIENTE, DATA, STATUS, PREV_DEVOLUCAO) '+
    'VALUES (?, ?, ?, ?, ?)';
  { Criando os parâmetros }
  prVideo   := FbParameter.Create;
  prCliente := FbParameter.Create;
  prData    := FbParameter.Create;
  prStatus  := FbParameter.Create;
  prDev     := FbParameter.Create;
  { Adicionando os parâmetros }
  DataAdapter.InsertCommand.Parameters.Add(prVideo);
  DataAdapter.InsertCommand.Parameters.Add(prCliente);
  DataAdapter.InsertCommand.Parameters.Add(prData);
  DataAdapter.InsertCommand.Parameters.Add(prStatus);
  DataAdapter.InsertCommand.Parameters.Add(prDev);
  { atribuindo valores ao parâmetros }
  DataAdapter.InsertCommand.Parameters[0].Value := ddlVideos.SelectedValue;
  DataAdapter.InsertCommand.Parameters[1].Value := ddlClientes.SelectedValue;
  DataAdapter.InsertCommand.Parameters[2].Value := txtData.Text.Replace('/','.');
  DataAdapter.InsertCommand.Parameters[3].Value := 'N';
  DataAdapter.InsertCommand.Parameters[4].Value := TxtPrevDev.Text.Replace('/','.');;
  if DataAdapter.InsertCommand.ExecuteNonQuery > 0 then
  begin
    CarregaLocacoes;
    CarregaVideos;
  end;
end;
```

Registrada a nova locação, serão chamados novamente os procedimentos para carregar Vídeos e preencher a lista de Locações ativas.

Para a devolução de locações ativas utilizaremos o *link* criado no *gridLocacao* e a codificação necessária será realizada no evento *ItemCommand* do mesmo, onde iremos instanciar um procedimento a ser criado. Declare também na seção *Private* a *procedure* a seguir:

```
procedure DevolverVideo(id_locacao: String)
```

Que irá receber como parâmetro o código da locação a ser devolvida. Utilizando o atalho *Ctrl+Shift+C* crie o cabeçalho e adicione o código da **Listagem 6** onde estamos criando os componentes em tempo de execução e atribuindo ao objeto *UpdateCommand* do *DataAdapter* a instrução SQL para atualização do *status* do registro de acordo com o *id* da locação que é passado pelo parâmetro do procedimento.

Na devolução do vídeo, simplesmente atualizamos a tabela de locação passando para ela a informação que o vídeo foi entregue, nesse caso através do campo *Status*.

A chamada do método acontece no evento *ItemCommand* do *gridLocacao* e será necessário adicionar a seguinte linha de código, onde o parâmetro é obtido pelo valor da primeira coluna na linha selecionada. No próprio método, após executado a página será recarregada e as informações serão atualizadas:

```
DevolverVideo(e.Item.Cells[0].Text);
```

Adicione ainda ao *btnVoltar* o código seguinte:

```
Response.Redirect('principal.aspx');
```

Isso irá redirecionar o administrador de volta à página principal. Nela adicione ao evento *Click* do *btnLocacao* o código uma nova chamada ao método *Redirect* do objeto *Response*, desta vez passando o arquivo *a_locacao.aspx*. Execute seu sistema, onde agora estará disponível as opções de criar novas reservas, e realizar a devolução de reservas ativas.

Conclusão

Na conclusão deste artigo e da parte anterior vimos as etapas mais importan-

Listagem 6. Devolução de locações

```
procedure TWebForm1.DevolverVideo(id_locacao: String);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prStatus : FbParameter;
  prDataFinal : FbParameter;
  prCodLocacao : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;

  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;

  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText := 'UPDATE LOCACAO SET STATUS = ?, '+
   'DATA_FINAL = ? WHERE (COD_LOCACAO = ?)';
  prStatus     := FbParameter.Create;
  prDataFinal  := FbParameter.Create;
  prCodLocacao := FbParameter.Create;
  DataAdapter.UpdateCommand.Parameters.Add(prStatus);
  DataAdapter.UpdateCommand.Parameters.Add(prDataFinal);
  DataAdapter.UpdateCommand.Parameters.Add(prCodLocacao);
  DataAdapter.UpdateCommand.Parameters[0].Value := 'D';
  DataAdapter.UpdateCommand.Parameters[1].Value := DateTime.Today.ToString('dd.MM.yyyy');
  DataAdapter.UpdateCommand.Parameters[2].Value := id_locacao;

  if DataAdapter.UpdateCommand.ExecuteNonQuery > 0 then
  begin
    CarregaLocacoes;
    CarregaVideos;
  end;
end;
```

te para a criação de um sistema *Online* para vídeo locadora. Codificamos as principais funcionalidades e métodos que são utilizados no sistema. Devido ao pouco espaço disponível para a criação do artigo, algumas partes foram resumidas, mas você poderá aumentar seus conhecimentos ampliando as páginas com funções de pesquisas e novos cadastros, como por exemplo a manutenção de clientes e usuários. Faça também uma verificação para atrasos na devolução de locações e aplique multas para estes casos. Na página do cliente, crie relatórios dinâmicos para atrair o interesse do cliente em ficar ativo no sistema, tornando isso como um diferencial do sistema para aumentar seu negócio. Grande abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Utilizando Threads na Prática

Veja todos os detalhes e dicas de como e onde utilizar Threads em suas aplicações

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atual como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

O foco de nosso artigo são as *Threads*. Mas, o que são *Threads*? Veremos nesse artigo, muito além do que elas são, ou fazem, pois abordaremos a sua utilização prática. Conheceremos o conceito, analisaremos a classe *TThread* e criaremos um exemplo para aplicarmos a teoria. É importante, sempre que possível, abstrairmos os conceitos e funcionalidades das tecnologias, para a realidade, seja ela o desenvolvimento do nosso software, ou do software da empresa onde trabalhamos.

O que são Threads?

TThread é uma classe abstrata e uma classe abstrata é sempre uma superclasse que não possui instâncias. Ela define um modelo implementando métodos abstratos, ou seja, sem implementação para que possamos utilizá-lo da maneira que precisarmos. A classe *TThread*, possibilita a criação de *Threads* separadas, para utilizarmos em nossas aplicações. Simplificando, *threads* são similares a processos, que são executados em *background*, ("por trás da aplicação"). Podemos utilizar o seguinte exemplo para entendermos melhor:

- Imaginemos que precisamos abrir uma tabela para manipularmos alguns registros, mas não temos outra opção, a não ser abrir toda a tabela. Essa operação de chamar o método *Open* da classe *TDataSet* ou "setar" a propriedade *Active* para *True*, carrega todos os registros da tabela em memória. Tranqüilo? Sim se a tabela for pequena, mas imaginemos uma tabela com 1.000.000 de registros, com certeza levará um tempo para que o processador consiga ler essas informações no HD e carregá-las na memória. Esse processo também é conhecido por *overhead*. Enquanto o processo está em andamento nosso sistema "trava" aos olhos do usuário. Nem tente explicar para ele, que você esta executando uma operação que exige muito do processa-

dor e que, por isso, o sistema pode ficar lento por uns momentos. Com o passar do tempo, e com o constante crescimento da concorrência no mercado de trabalho, temos mais e mais tarefas para serem executadas.

É completamente inaceitável para uma empresa, depender de um software que "trava" sempre que uma tabela esta sendo aberta, ou seja, carregada para a memória. Essa é uma das situações em que as *Threads* nos ajudam, pois se a utilizarmos para carregar os registros em memória o sistema não vai travar e o usuário pode continuar seu trabalho, sem perceber o que esta por trás ("background").

Conhecendo a classe TThread

A classe *TThread* é declarada na *Unit Classes* do Delphi. E ela possui algumas características ("métodos") que precisam de atenção para que possamos entender perfeitamente. Vejamos algumas delas:

• *Execute:* Método abstrato. Esse método é sempre sobrescrito ao "desenharmos" nossas próprias *threads*. É ele que efetivamente põe a *thread* e execução;

• *Synchronize:* Responsável pelo *Sincronismo*, entre as *thread's* que estiverem sendo executadas, incluindo as *thread's* do sistema operacional;

• *Suspend:* Suspende ("pausa") temporariamente a execução;

• *Resume:* Reinicia a execução de um processo suspenso;

• *Terminate:* Seta propriedade *Terminated* para *True*;

• *WaitFor:* Aguarda o término da *thread* e retorna o valor da propriedade *ReturnValue*;

• *Priority:* Indica a prioridade da *thread* em relação às outras *thread's* que estão em execução.

Criando uma aplicação

A aplicação que criaremos será simples, mas suficiente para entendermos onde e quando devemos utilizar *Thread's*. Vamos utilizar o Delphi 7 para criarmos nosso exemplo. A idéia inicial que proponho é criarmos uma aplicação que precisa acessar uma tabela para poder alterar os dados da mesma. A base de dados que utilizarei em nosso exemplo, é o SQL Server Express, que é a versão *free* do SQL Server. Nele vamos criar uma tabela e salvá-la com o nome de Clientes. Após isso, vamos preenchê-la com 1.000.000 (um milhão) de registros. Isso será mais do que suficiente, para demonstrarmos o tempo que o sistema levará, para abrir a tabela, sempre que necessário.

Não vou entrar em detalhes quanto a criação da base de dados e da tabela, por não ser o foco do artigo. Não é necessário criar a tabela no SQL Server. Você pode utilizar qualquer banco de dados que estiver habituado a trabalhar, tais como Firebird, MySQL. O método que criaremos para preenchê-la funciona da mesma forma com qualquer base de dados. A tabela deve possuir apenas um campo inteiro e outro *Varchar*(50).

Salvamos a tabela com o nome de *Clientes*, feito isso, vamos codificar o botão para preencher os registros como proposto. A interface do sistema pode ser simples, como na **Figura 1**.

Os componentes necessários para criar a interface da aplicação de exemplo são:

• *ProgressBar*: para que possamos acompanhar o preenchimento da tabela;

• *DbGrid*: para podermos visualizar e alterar os dados;

• *ADOTable*("TblClientes"): Como utilizei o SQL Server para criar a tabela, utilizei um *ADOTable* para fazer a conexão com a base e manipular a tabela;

• 2 *Buttons:* um para abrir a tabela, e outro para gravar os registros no evento *OnClick*.

Listagem 1. Código para gravação de registros

```
procedure TfrmThreads.btnGravarClick(Sender: TObject);
var
 I : Integer;
begin
  { Passo o valor máximo de registros para a propriedade Max do ProgressBar }
  ProgressBar1.Max:= 1000000;

  { Chamo o método Open do DataSet }
  TblClientes.Open;

  { Mostro uma mensagem com a quantidade de registros existente na tabela de clientes }
  ShowMessage(IntToStr(TblClientes.RecordCount));

  { Desabilito o controle visual chamando o método DisabelControls. Esse método é muito importante
    para melhorar a performance da aplicação, deve ser utilizando sempre que necessário
    percorrer os registros de um DataSet }
  TblClientes.DisableControls;

  { Passo os valor para o laço }
  for i:= 0 to 1000000 do
  begin
    { Chamo o método insert do DataSet para colocar a tabela em modo de inserção }
    TblClientes.Insert;

    { Passo um valor para o Field Nome da tabela de clientes }
    TblClientesNome.AsString := 'Sou o cliente: ' +
      IntToStr(i);

    { Chamo o método Post para gravar os registros }
    TblClientes.post;

    { Atualizo o progresso do ProgressBar }
    ProgressBar1.Position := i;
  end;

  { No final do método mostro uma mensagem informando que os registros foram gravados com sucesso }
  ShowMessage('Todos os clientes gravados com sucesso!');
end;
```

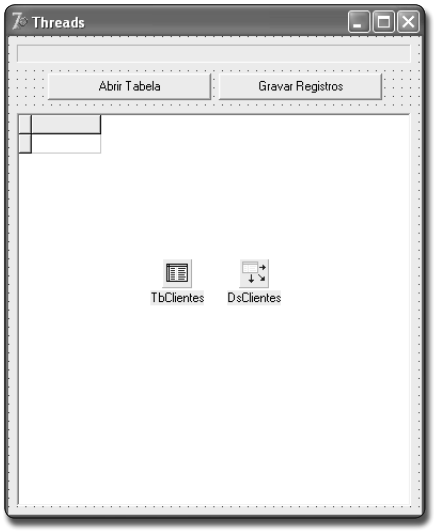


Figura 1. Janela de exemplo

Faremos um laço para preencher a tabela e o botão de abrir tabela, deve chamar o método *Open* da classe *TDataSet* ou *setar* a propriedade *Active* para *True*. Na **Listagem 1** segue o código do botão gravar registros. Todo o código encontra-se comentado para melhor entendimento.

Após montar o exemplo, execute-o e clique em *Gravar Registros*. Executado esse procedimento, temos 1.000.000 de registros gravados na tabela de clientes. Agora poderemos iniciar os testes. O primeiro teste que faremos será abrir a tabela e marcar o tempo em segundos, que o sistema levará para carregar todos os registros para a memória e liberar a tela da aplicação. Notaremos que a tela ficará travada, até que o *Open* seja finalizado. Antes de executarmos a aplicação, vamos codificar o evento *BeforeOpen* e *AfterOpen* do componente *Table*. Os mesmo serão executados antes e depois da tabela processar o *Open*, respectivamente. A **Listagem 2** mostra o código de ambos eventos.

Nota: Declare as variáveis *HoraFim* e *HoraIni*, ambas do tipo *TDateTime*, na seção *public* do formulário.

No botão *Abrir Tabela* adicione o código da **Listagem 3**. O código é bem simples, apenas invocamos o método *Open* do objeto *AdoTable*, subtraímos a *HoraFim* da *HoraIni* e então mostramos uma mensagem com o tempo em segundos que a tabela levou para ser aberta. (**Figura 2**)

Perceba, pelas imagens, que em minha máquina o tempo para abertura da tabela foi de 51 segundos. Isso significa que foram 51 segundos com o sistema "travado". É claro que esse valor pode variar ainda mais dependendo da máquina onde se executa o procedimento, dos processos que estão sendo executados, tráfego na rede e uma série de outras situações corriqueiras em uma rede.

Agora imaginemos um usuário com pilhas de dados a serem alterados, esperar pacientemente até que a tabela esteja disponível para alteração. No mínimo ele já terá clicado em *Fechar* ou *CTRL+ALT+DEL*, pensando que o sistema está trancado. Nesse exemplo apresentamos uma situação problemática, agora cabe a nós resolvermos. Uma das soluções para essa situação é utilizarmos uma *Thread*, e dispararmos o método *Open* da tabela dentro dessa *Thread*.

Listagem2. Código dos eventos BeforeOpen e AfterOpen

```
procedure TForm1.TblClientesAfterOpen(DataSet: TDataSet);
begin
  ShowMessage(TimeToStr(Now));
  HoraFim := Now;
end;

procedure TForm1.TblClientesBeforeOpen(DataSet: TDataSet);
var

begin
 ShowMessage(TimeToStr(Now));
 HoraIni := Now;
end;
```

Listagem 3. Código do botão Abrir Tabela

```
procedure TThreads.btnAbrirTabelaClick(Sender: TObject);
var
  TempoTotal : TDateTime;
begin
  { Chamo o método Open do DataSet }
  TblClientes.Open;

  { Calculo o tempo utilizado  }
  TempoTotal := HoraFim - HoraIni;

  { Mostro uma mensagem com o tempo total que o sistema levou para executar o Open}
  ShowMessage('Tempo total em segundos para abrir a tabela: ' + ' - ' +
    TimeToStr(TempoTotal));
end;
```

Criando uma Thread

Para criamos uma *Thread* acesse o menu *File|New|Other|New|Thread Object*. Clicando em *Ok*, será criada uma nova *Unit*. Digite o nome da classe em *Class Name* ("TBackGround") e salve a Unit como "uBackGround.pas". Declare o método *AbreTabela* na seção *Private* como segue:

```
...
  private
    { Private declarations }
    procedure AbreTabela;
  protected
    procedure Execute; override;
  end;
...
```

Em seguida pressione *CTRL + SHIFT + C* para que o Delphi nos crie o cabeçalho do método. Encontre sua implementação na seção *Implementation* da *Unit* e digite o código a seguir:

```
Form1.TblClientes.Open;
```

Na seqüência sobrescreva o método *Execute* com o apenas invocando o método *Synchronize* como segue:

```
Synchronize(AbreTabela);
```

Como vimos na explicação teórica sobre *Thread*, devemos utilizá-la criando uma classe descendente de *Thread* e sobrescrever o método *Execute*. Todo esse trabalho de criar a estrutura da classe com a declaração do método *Execute* foi feito pelo Delphi ao adicionamos o *Thread Object* na aplicação. O próximo passo, foi criarmos uma método chamado *AbreTabela*, que executará o método *Open* da *Table*. Como *TblClientes* está na *Unit* principal, devemos adicioná-la na *Uses* de *uBackGround*, ou seja, pressione *Alt + F11* e inclua *uPrincipal* ao *Uses*.

O método *Synchronize* é utilizado para executar o método *AbreTabela*. Este, é responsável por fazer sincronismo entre as *Threads* da VCL e de nossa aplicação. Essa é a maneira correta de chamarmos os métodos dentro do método *Execute* da *Thread*.

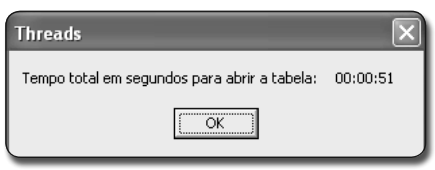


Figura 2. Tempo para abrir a tabela

Voltando ao formulário principal de nossa aplicação, faremos as alterações necessárias para utilizarmos a *Thread*. O primeiro passo é adicionarmos o *uBackGround* na *Uses* do formulário. Feito isso, ao invés de chamar-mos o método *Open*, no evento *OnClick* do botão de *Abrir Tabela*, vamos executar a nossa *Thread* criando uma instância dela. Modifique o código do botão *Abrir Tabela* pelo código que vemos a seguir:

```
TBackGround.Create(False);
```

O esquema é bem simples, basta criarmos a *Thread* e passarmos como parâmetro *False*. Esse parâmetro indica que a *Thread* deve ser criada e executada imediatamente. Testando a aplicação podemos perceber que mesmo sendo uma tabela com centenas de registros a aplicação não trava e podemos continuar interagindo com o sistema.

Monitorando a Execução da Thread

Pode ser necessário monitorarmos a execução da *Thread* no momento em que ela é instanciada e executada. Para isso devemos colocar um *BreakPoint* no evento *OnClick* do botão *Abrir Tabela* e acessarmos o menu *View|Debuge Windows>Threads*. Uma janela semelhante a **Figura 3** será mostrada. Nela temos o *ID*, ou seja, o identificador de todas as *Thread's* que estiverem em execução no momento. O *State* ("estado") da mesma e também a localização dela em memória.

Nota: É importante considerarmos, que as *Threads* são executadas diretamente no processador, o que pode comprometer o processamento caso abusarmos de mais do uso das mesmas. Atualmente, os computadores possuem processadores com grande capacidade de processamento, mas devemos ficar atentos caso nossa aplicação precise rodar em máquinas mais lentas.

Conclusão

Nesse artigo, conhecemos e criamos um exemplo prático de utilização de *Threads*, uma poderoso recurso disponível no Delphi. Não devemos nos esquecer que ele é muito útil, e também pode ser utilizado para gerar grandes relatórios que envolvam *Joins* em várias tabelas, ou ainda atualizarmos o estoque de produtos após a digitação e ou impressão de vários pedidos. As situações para o uso de *Thread* podem ser as mais variadas possíveis. Conhecendo e entendendo os conceitos, basta utilizarmos a imaginação para melhorarmos a performance da nossa aplicação. Bons códigos e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

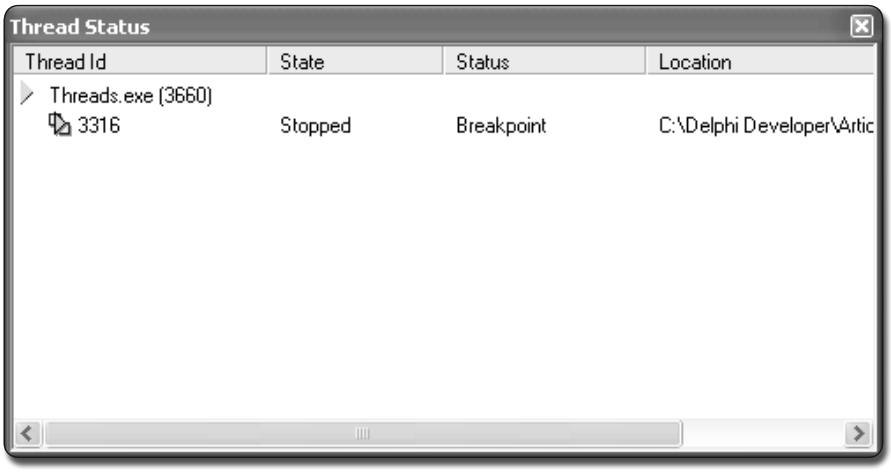


Figura 3. Threads em execução na aplicação

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Stored Procedures

Aprenda a criar e utilizar Stored Procedures em suas aplicações Win32 e .NET

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Se traduzirmos ao pé da letra o termo *Stored Procedure*, chegaremos em "Procedimentos Armazenados". Podemos chamar também de *SP's*. *SP's* são um conjunto de comandos que atribuímos uma identificação ("nome") e deixamos armazenado no banco de dados, sendo que a qualquer momento este procedimento poderá ser solicitado por meio do SGBD ("Sistema Gerenciador de Banco de Dados") ou através de um sistema específico com interface e integração com a base de dados. A idéia que se tem quando falamos em *Stored Procedures* é a criação de um sistema dentro do próprio banco, onde todo o processo é executado podendo ser definidas regras para cadastros, alterações, exclusões, entre outras.

Veremos neste artigo os conceitos e dicas de como e o porquê de usar procedimentos armazenados, criação e utilização de *Stored Procedures* através do Delphi 7 para sistemas *Win32* e do Delphi 2006 *for .NET* para aplicações *Web for ASP.NET*. Faremos a implementação da ferramenta na base de dados *Employee.fdb* que acompanha a instalação padrão do Firebird e utilizaremos o IBExpert para criação de tais SP's.

Conceito

Coleção de comandos em SQL para gerenciamento da base de dados, as *Stored Procedures*, permitem ao desenvolvedor definir rotinas e tarefas encapsuladas a serem executadas de forma repetitiva na base recebendo como informações os parâmetros de entrada (*in*) e retornando valores de resposta ao usuário, conhecidos como parâmetros de saída (*out*). Os procedimentos criados em um banco poderão ser executados a qualquer momento. Eles podem ser solicitados por aplicações ou quando determinada condição é atendida, mesmo que não haja inter-

venção de usuários. Podendo também ser anulada qualquer taxa de tráfego de informações pela rede visando que os registros e todo o processamento serão de forma local na estação onde o banco de dados encontra-se hospedado.

Em outras palavras, uma *Stored Procedure* pode ser executada diretamente da aplicação, seja ela *Client/Server, n-tier, Web* etc., ou ainda dentro do próprio SGBD através de outras *SP's* ou gatilhos, chamados também de *Trigger's*.

Aspectos gerais e vantagens de utilizar Stored Procedures

A utilização de *SP's* faz com que seu sistema ganhe performance profissional, maior rendimento durante o desenvolvimento e possibilita até correção de falhas. Conheça algumas das vantagens em se utilizar *SP's*. Antes de desenvolver um sistema faça análise de toda a estrutura e ações necessárias baseando-se nos itens a seguir. Faça o levantamento de quando e como utilizar *Stored Procedures*:

• *Mais velocidade e desempenho:* Ações como alteração, inclusão e consultas de informações são muito mais rápidas quando realizadas através de *Stored Procedures;*

• *Processamento direcionado: Stored Procedures* "rodam" diretamente no servidor, aspecto este que possibilita atualizações sem preocupação com falhas nos aplicativos desde que seus parâmetros não sejam alterados;

• *Redução do tráfego:* Apenas a chamada e os parâmetros serão transferidos pela rede;

• *Operações complexas:* podemos executar processos bastante complexos e extensos sem que haja interatividade com aplicativos externos. Dessa forma podemos estender todo o poder da linguagem SQL fazendo o banco trabalhar por nós;

• *Divisão de tarefas:* enquanto a requisição de um procedimento realizado por um usuário é processada no servidor o mesmo poderá continuar a executar outras atividades no aplicativo *cliente*;

• *Obtenção de resultados:* em uma única chamada a um procedimento é possível passar parâmetros para a base de dados e obter resultados das ações executadas ou pesquisas realizadas.

Em determinadas circunstâncias, o uso de procedimentos é a forma mais aconselhável. Podemos citar como situações em potencial para o uso de *SP's*: operações que não necessitem da intervenção do usuário, processos iguais sendo executados por mais de um dos módulos ou usuários em diferentes terminais, ações diárias ou repetitivas, necessidade de grande processamento ou número elevado de registros resultantes, entre outras situações. Ou seja, regras e comandos SQL ficam centralizados no aplicativo, o que nem sempre é o ideal.

Criando o primeiro procedimento

A primeira implementação de uma *Stored Procedure* será realizada com auxilio do IBExpert em sua versão *Standard*. Por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de *download* da ferramenta. Após isso, abra o IBExpert e vamos registrar o banco de dados *Employee.fdb* através do menu *DataBase>Register DataBase*. Na janela de registro (**Figura 1**) altere a caixa de seleção *Server* para "*Local*" e *Server Version* para "Firebird 1.5". Em *Database File* localize o banco de dados *Employee.fdb* encontrado no caminho *default* de instalação do Firebird no endereço "*C:\Arquivos de programas\Firebird\Firebird<versão>\examples\empbuild\EMPLOYEE.FDB*". Em *Database Alias* informe um "apelido" que deseja atribuir ao registro da base ("EMPLOYEE"). Informe ainda o *User Name* "SYSDBA" e *Password* "masterkey" para autenticação na base de dados. Confirme a janela de configurações e localize o apelido no *DB Explorer* ("F11") e ative-o com um duplo clique, visualizando assim a estrutura do banco.

Finalizado o registro da base de dados, abra no menu *Tools|SQL Editor*. No editor de comandos SQL deverá permanecer selecionada a opção *"Use current connect"* e digite a SQL da

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

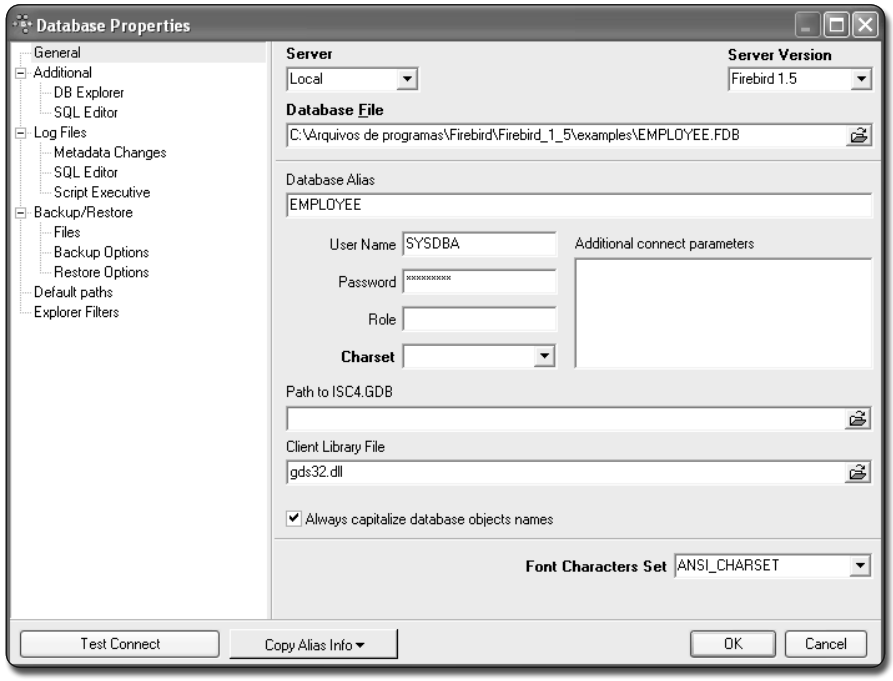


Figura 1. Registrando a base de dados

Listagem 1. Criando procedimento de pesquisa

```
/* Declaração da Stored Procedure e seu parâmetro de
   entrada (Código) */
CREATE PROCEDURE Proc_Employee(CODIGO INTEGER)
/* Parâmetros de saída (Return) */
RETURNS(
  EMP_NO INTEGER,
  FIRST_NAME VARCHAR(15),
  LAST_NAME VARCHAR(20),
  SALARIO NUMERIC(10,2)
  )
AS
BEGIN
  SELECT
    EMPLOYEE.EMP_NO, EMPLOYEE.FIRST_NAME,
    EMPLOYEE.LAST_NAME, EMPLOYEE.SALARY
  FROM
    EMPLOYEE
  WHERE
    /* Instrução SQL que fará a pesquisa */
    (EMPLOYEE.EMP_NO = :codigo)
    /* Variáveis de saída recebendo o resultado */
    INTO :EMP_NO, :FIRST_NAME, :LAST_NAME, :SALARIO;
  Suspend;
END
```

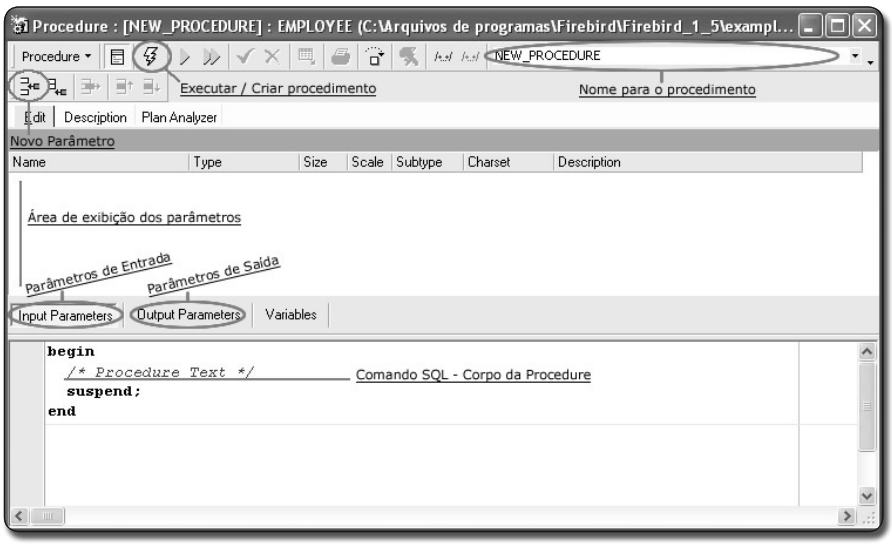


Figura 2. Criando um novo procedimento

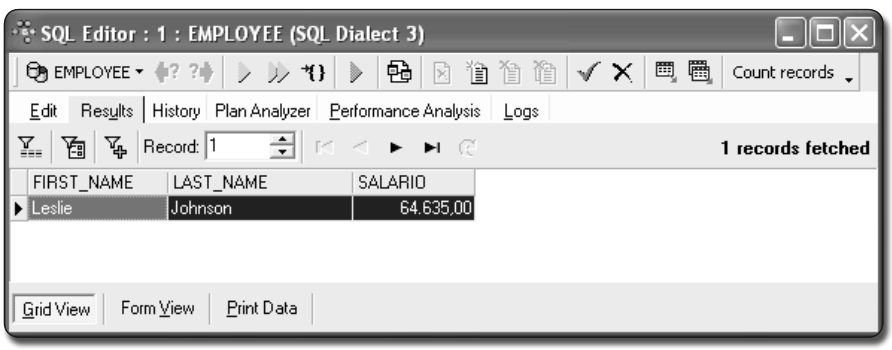


Figura 3. Exibição do resultado do procedimento

Listagem 1 que será responsável pela criação do procedimento "Proc_Employee" destinado a pesquisar algumas informações de acordo com o código informado através do parâmetro de entrada "CODIGO". As demais linhas da instrução encontram-se comentadas de acordo com a ação a ser realizada.

Depois de digitado o código, utilize o botão *Run Script* ("F9") para executar o código de criação do procedimento e confirme usando o botão *Commit Transaction* ou pressione *Ctrl + Alt + D*. A confirmação de que o procedimento foi criado poderá ser vista observando-se a categoria *Procedures* no *Database Explorer*.

Utilizando o IBExpert, a criação de um procedimento poderá também ser resumida a um *Wizard* de criação fornecido pela própria ferramenta. Através do menu *Database>New Procedure*, uma janela (**Figura 2**) irá disponibilizar todas as opções necessárias para criação e configuração do procedimento, tais como: inclusão de parâmetros de entrada, parâmetros de saída e variáveis, codificação do corpo da *Stored Procedure*, descrição do procedimento, ferramentas para inserção ou exclusão de trechos comentados no decorrer do código além de ferramentas para *Debug* entre outras funcionalidades.

Durante a inclusão de parâmetros, é muito importante observar a configuração do tipo de parâmetro a ser criado. Parâmetros para a obtenção de textos, deverão ser configurados como *Type Varchar* e em *Size* com a quantidade de caracteres válidos a serem aceitos durante a execução. Um parâmetro que receberá números deverá ser configurado como *Integer*. Para valores monetários podemos utilizar *Numeric(12,2)* onde define-se o tamanho de espaço disponível para números e a quantidade de casas decimais após a vírgula. Enfim, cada campo deve ser definido de acordo com a sua necessidade.

A execução de procedimentos poderá ser realizada também pelo IBExpert através do editor de comandos SQL em *Tools|SQL Editor*, onde iremos chamar o procedimento já criado *Proc_Employee* digitando o seguinte código:

```
SELECT * FROM PROC_EMPLOYEE(8)
```

O código anterior é muito semelhante a um *Select* normal em tabelas da base dados, a diferença é que passamos um valor ao parâmetro de entrada do procedimento logo após referenciar seu nome, entre parênteses, não precisando neste caso da inclusão de condições através do *Where*. A utilização do asterisco indica que desejamos mostrar no resultado todos os parâmetros de saída configurados no procedimento. Para estabelecer limites ou definir quais campos estariam visíveis no resultado, poderíamos simplesmente substituir o asterisco pelo nome dos campos a serem exibidos, deixando o código SQL da seguinte forma:

```
SELECT FIRST_NAME, LAST_NAME, SALARIO FROM
PROC_EMPLOYEE(8)
```

Após executar a consulta, serão exibidos os resultados (**Figura 3**) de acordo com o parâmetro repassado ao procedimento, atendendo a condição *Where* anteriormente adicionada à instrução SQL.

Usando Stored Procedures em aplicações Win32

A utilização de *Stored Procedures* em aplicações traz inúmeros benefícios como já vimos anteriormente. O uso de procedimentos durante o desenvolvimento é bastante comum e simples de ser implementado, dispensando comandos extensos ou complicados. Iremos criar um novo procedimento através do IBExpert e logo após passar a utilizá-lo em uma aplicação criada no Delphi 7. Para realizar a conexão com a base de dados e executar o procedimento. Iremos utilizar os componentes da paleta *dbExpress* por padrão já instalados no Delphi.

Figura 4. Criando uma nova conexão

Para a aplicação exemplo, criaremos um procedimento que fará a inclusão de um registro na tabela de clientes ("CUSTOMER") e ao mesmo tempo retornará o código do cliente cadastrado. Código este que para esta tabela se trata de um campo auto-incremento e precisamos fazer um *Select* no *Generator* para descobrir seu valor atual.

> **Nota**: O banco de dados Firebird possui um recurso chamado *Generator* que faz a geração automática de números para campos do tipo auto-incremento. São muito usados para campos de código que não podem se repetir.

O procedimento a ser criado contará tanto com parâmetros de entrada quanto com parâmetros de saída. Sua criação é feita a partir do executor de comandos SQL em *Tools|Script Executive*. Digite o código da **Listagem 2** que encontra-se comentado de acordo com a ação de cada linha a ser executada e logo após utilize o botão *Run Script (F9)* para compilar e criar o procedimento. Finalizado com êxito, nosso procedimento já estará pronto para ser utilizado.

Concluída a criação do procedimento *Ins_Customer* para cadastro de novos clientes, abra o Delphi 7 e no menu *File|New>Application* crie uma nova aplicação. Altere a propriedade *Caption* do formulário principal para "Cadastro de Clientes com *Stored Procedures*" e *Name* para "frmCustomer". Salve a *Unit* principal do projeto como "uCustomer.pas" e o projeto como "prjCustomer.dpr". Adicione ao formulário, da paleta *dbExpress*, os componentes responsáveis pela conexão com a base de dados e execução da *Stored Procedure*. Para configurar a conexão adicione o componente *SQLConnection* ("sqlConexao") e com um duplo clique sobre o componente crie uma nova conexão utilizando o ícone *Add Connection*, definindo o *Driver Name* para "Interbase" e *Connection Name* para "Employee" (**Figura 4**). Confirme e prossiga para a configuração dos demais parâmetros, começando pela propriedade *Database* onde deverá ser informado o caminho da base de dados *Employee.fdb* (<caminho>\employee.fdb). Defina as credenciais para acesso ao Firebird, usuário "SYSDBA" e senha "masterkey".

Na propriedade *ServerCharSet* utilizaremos o padrão *Win1252* e em *SQLDialect* utilize o padrão 3 para que possamos utilizar comandos SQL mais complexos (**Figura 5**).

Ainda no componente *sqlConexao*, defina sua propriedade *LoginPrompt* para *False* e

Listagem 2. Criando procedimento de inclusão de registros

```
/* Parâmetro de entrada */
CREATE PROCEDURE Ins_Customer(
  CLIENTE varchar(25),
  CONTACT_FIRST varchar(15),
  CONTACT_LAST varchar(20),
  PHONE_NO varchar(20),
  ADDRESS_LINE1 varchar(30),
  PAIS varchar(15))
/* Parâmetro de saída */
RETURNS (
  CODIGO Integer
  )
AS
BEGIN
  /* Código de inserção para cadastro de novo
    cliente
  */
  INSERT INTO CUSTOMER (
    CUSTOMER, CONTACT_FIRST,
    CONTACT_LAST, PHONE_NO, ADDRESS_LINE1, COUNTRY)
  VALUES (:CLIENTE, :CONTACT_FIRST, :CONTACT_LAST,
    :PHONE_NO, :ADDRESS_LINE1, :PAIS);
    /* Código responsável por pesquisar o código do
      último cliente cadastrado. CUST_NO_GEN é o nome
      do generator criado para o primary key da tabela
      CUSTOMER
    */
    SELECT
      gen_id(CUST_NO_GEN,0)
    FROM
      RDB$DATABASE
    INTO
      :CODIGO;
  Suspend;
END
```

altere a propriedade *Connected* para *True* testando assim o sucesso da conexão.

Para chamar e executar a *SP*, adicione, também da paleta *dbExpress*, o componente *SQLStoredProc* ("SQLStoredProc1") e relacione sua propriedade *SQLConnection* para o componente *sqlConexao*. Na propriedade *StoreProcName* selecione *Ins_Customer*. Observe que na propriedade *Params* do componente já foram adicionados, automaticamente, os parâmetros que definimos durante sua criação no IBExpert. Para visualizá-los basta dar um duplo clique em *Params*, onde veremos, já se encontram devidamente configurados os parâmetros para entrada de dados ("ptInput") e também os parâmetros de retorno de dados ("ptOutput").

Para deixar o cadastro de novos clientes de uma forma mais dinâmica, criaremos um pequeno formulário de cadastro que deverá atender o preenchimento de todos os parâmetros necessários. Desenhe uma tela como na **Figura 6**.

Adicione seis componentes *Edit* com os nomes "edtCliente", "edtContactFirst", "edtContactLast", "edtFone", "edtEndereco" e "edtPais". Limpe a propriedade *Text* de todos eles e coloque um *Label* acima de cada *Edit* com a descrição do campo. Para finalizar o formulário de cadastro, adicione um *Button*("btnCadastrar") alterando o seu *Caption* para "Cadastrar Cliente".

A codificação de inserção das informações na base de dados através de *Stored Procedures* é bastante simples. Uma vez que todos o parâmetros já se encontram automaticamente configurados, basta atribuir valores a eles e executar o procedimento. Após a execução do procedimento, os parâmetros de retorno serão carregados, onde poderemos trabalhar com os valores recebidos. Para entender melhor o cadastro de dados utilizando *Stored Procedures*, adicione ao evento *OnClick* do *btnCadastrar* o código da **Listagem 3**.

Perceba que o código é visivelmente simples. Envolvi o componente *SQLStoredProc1* em um bloco *with..do* para facilitar a passagem de parâmetros. Alimentei cada parâmetro como valor dos seus respectivos *Edit's* e então chamei o método *ExecProc* do componente. Além de envolver o componente em um bloco with..do, envolvi todo o procedimento em um bloco *try..except* o que significa que se tudo ocorrer bem o próximo comando após o *ExecProc* é executado. Nesse caso mostramos uma mensagem informando ao usuário final que o cliente foi adicionado com êxito. Caso contrário, o código previsto no *except* será chamado mostrando ao usuário uma mensagem de erro.

Finalizada a codificação do botão, compile e execute a aplicação informando os dados para serem cadastrados. Tome cuidado quanto ao campo país, pois este é um campo com referência em outra tabela, ou seja, *foreign key* ("chave estrangeira") e, portanto, deverá ser digitada a sigla de um país da mesma forma como se encontra no cadastrado. Ao finalizar o preenchimento dos campos, clique sobre o botão cadastrar e obtenha a mensagem do cadastro com o código que o mesmo registro assumiu na base de dados (**Figura 7**).

Utilizando Stored Procedures em aplicações .NET

Sem dúvida uma das maiores vantagens do uso de *Stored Procedures* em aplicações *Web* é com o ganho de desempenho. Principalmente em aplicações .NET, o uso de procedimentos faz com que os processos de cadastros aconteçam de forma muito mais ágil fazendo com que as páginas sejam carregadas mais rapidamente durante

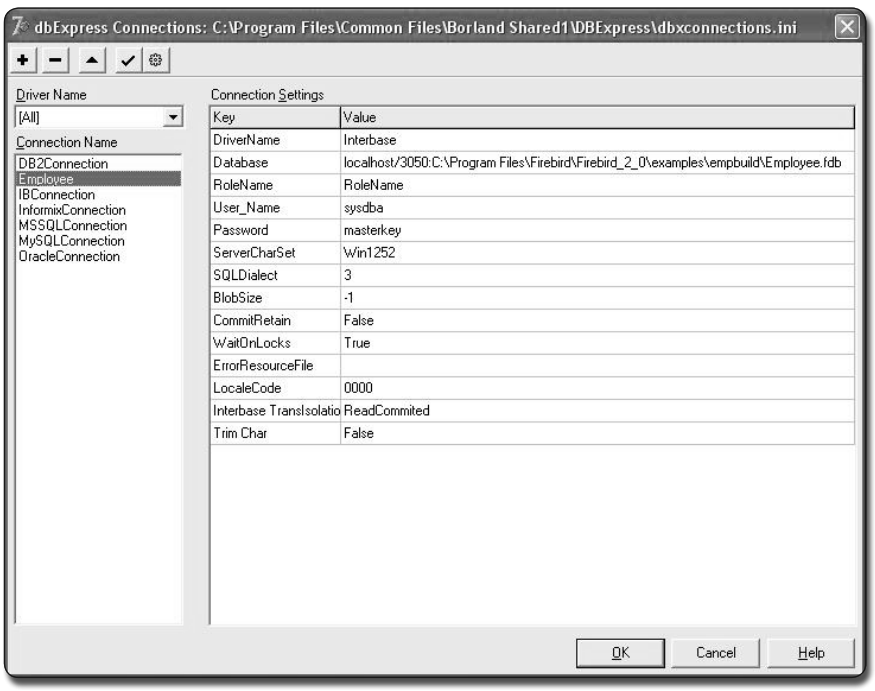


Figura 5. Configurando a conexão

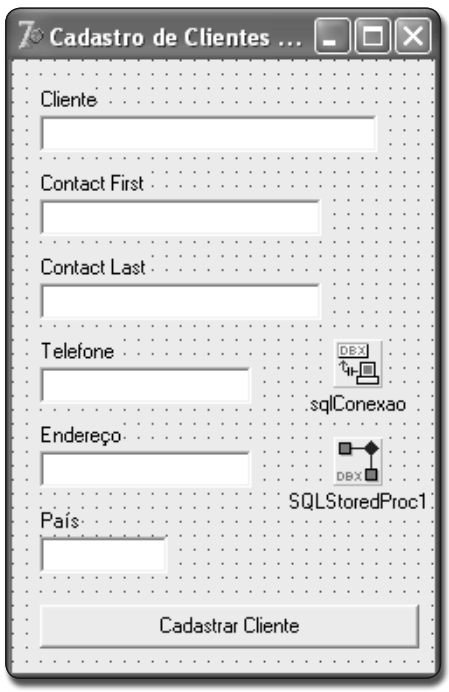


Figura 6. Exemplo de janela de clientes

o processo. Para ver na prática a utilização de *Stored Procedures* com .NET, abra o Delphi 2006 e crie um novo projeto no menu *File|New>ASP.NET Web Application – Delphi for .NET*. Na caixa de diálogo que aparece (**Figura 8**), defina o nome do projeto como "VideoLocadora", no item *Location* selecione o diretório destino onde os arquivos do projeto serão criados e no item *Server* o servidor de aplicação para a hospedagem do sistema.

Altere a página inicial, *WebForm1.aspx* para "principal.aspx", para isso clique com o botão direito do mouse no nome da página no *Project Manager* e selecione *Rename*. Adicione ao formulário uma tabela contendo 7 linhas e duas colunas, onde deverá adicionar e configurar os componentes conforme a seguir: adicione na linha 1 e coluna 1 o texto "Cliente:", na coluna ao lado da paleta *Web Controls* insira um *TextBox*("txtCliente"). Na linha 2 e coluna 1 digite o texto "Contato1" e insira um *TextBox*("txtContato1") na coluna ao lado. Na próxima linha adicione o texto "Contato2" e ao lado um *TextBox*("txtContato2"). Na linha seguinte o texto "Telefone" e na coluna ao lado um *TextBox*("txtTelefone"). Na linha abaixo o texto "Endereço" e o *TextBox*("txtEndereco") na coluna do lado e por último, para atribuir valor ao último parâmetro de entrada o texto "País" e no lado um *TextBox*("txtPais"). Na última linha da tabela, segunda coluna, adicione um *Button*("btnCadastrar") alterando sua propriedade *Text* para "Cadastrar Cliente" finalizando a montagem do *layout* do formulário de cadastro. Uma sugestão de layout pode ser vista na **Figura 9**.

Com um clique duplo sobre o botão *btnCadastrar,* faremos a implementação do seu evento *Click* no qual, de forma simples, iremos configurar uma conexão com a base de dados e por meio dos *providers* do Firebird .NET Provider (veja o box do DevMan), faremos o processo de conexão e cadastro de clientes utilizando *Stored Procedures* em tempo de execução. A primeira coisa que faremos é codificar o evento *Click* do botão, digitando o código da **Listagem 4.** Vamos às explicações para entendermos melhor.

Figura 7. Cadastro de cliente em aplicação Win32

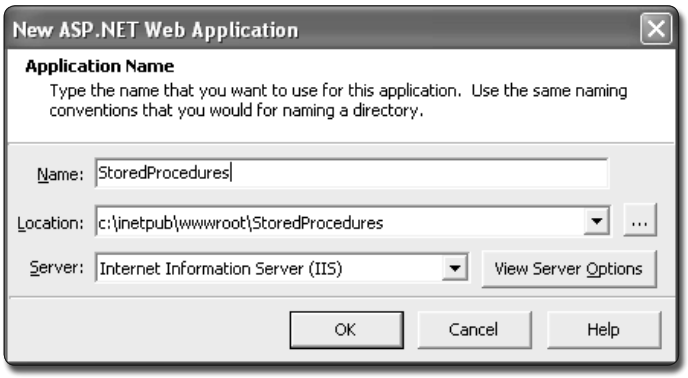


Figura 8. Criação da aplicação

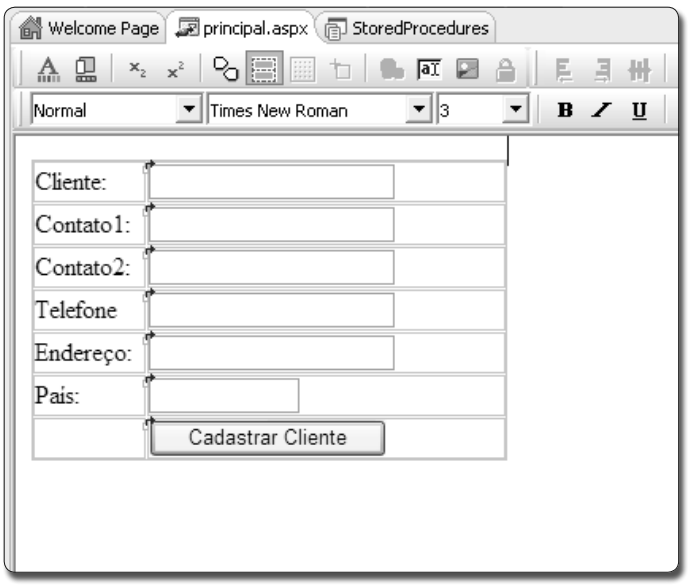


Figura 9. Formulário de cadastro em .NET

Nota do DevMan

Download e instalação do Firebird Data Provider

Acesse o endereço www.firebirdsql.com, clique no link Download e a seguir em Firebird .NET Data Provider. Baixe e instale a última versão do Data Provider for .NET Framework 1.1, bastando seguir os passos no assistente que será iniciado.

Nota: Neste artigo usaremos versão 1.7.1 do Provider, as demais versões estão disponíveis para Framework acima do 1.1 usado pelo BDS 2006.

Após a instalação, no Delphi 2006 clique no menu Component|Installed .NET Components. Digite "Firebird Data Provider" na opção Category, clique no botão Select an Assembly (**Figura 10**) e escolha o arquivo FirebirdSql.Data.Firebird.dll, localizado no diretório de instalação do Provider, por padrão em C:\Arquivos de programas\FirebirdNETProvider(versão). Clique em Ok e observe que os novos componentes para acesso ao Firebird estão agora disponíveis no IDE (**Figura 11**).

Visão geral dos componentes do Firebird Data Provider

A seguir conheceremos os componentes do Firebird Data Provider. Nem todos o componentes serão usados na prática durante o exemplo deste artigo, mesmo assim é importante conhecer desde já a função de cada um.

fbConnection – Sua função é estabelecer uma conexão com o BD.

Principais Propriedades:
- ConnectionString: contém as informações necessárias para conexão com o BD;

Principais Métodos:
- Open: Conecta ao banco;
- Close: Fecha a conexão.

fbCommand - Executa comandos SQL ou Stored Procedures.

Principais Propriedades:

- CommandType: permite especificar o tipo de comando, se é uma instrução SQL (Text), uma tabela (TableDirect) ou uma Stored Procedure (StoredProcedure);

- CommandText: especifica a instrução SQL que será executada, o nome da tabela ou Stored Procedure a ser acessada no banco (de acordo com o que foi escolhido na propriedade CommandType);

- Connection: a conexão que será usada pelo fbCommand.

Principais Métodos:

- ExecuteReader: usado para execução de SQLs ou Stored Procedures que retornam um cursor (por exemplo, comandos select). O retorno dessa função é um fbDataReader, objeto usado para fazer leitura unidirecional de dados;

- ExecuteNonQuery: usado para executar SQLs ou Stored Procedures que não retornam um cursor (por exemplo insert). O retorno dessa função é um integer, indicando o número de registros afetados;

- ExecuteScalar: retorna somente a primeira coluna do primeiro registro da consulta. Ideal e otimizado para consultas como Select Count, Select Max etc., já que para a leitura não é necessária a alocação de um fbDataReader, apenas um System.Object.

fbDataAdapter

Esse componente abstrai quatro fbCommands em uma única estrutura, através das propriedades SelectCommand, DeleteCommand, UpdateCommand e InsertCommand, todas do tipo fbCommand.

Principais Propriedades:
- SelectCommand: comando utilizado para obter os dados, que normalmente serão armazenados em um DataSet;
- DeleteCommand, UpdateCommand e InsertCommand: utilizados para fazer as atualizações das mudanças feitas no DataSet quando o método Update é chamado.

Principais Métodos:
- Update: faz a efetivação no BD das mudanças feitas no DataSet, utilizando os SQLs de Delete, Update e Insert das respectivas propriedades do fbDataAdapter;
- Fill: preenche um DataSet;

A **Figura 12** mostra a arquitetura de componentes do Firebird Data Provider.

Nota: O DataSet é um componente nativo do ADO.NET e não faz parte do Firebird Data Provider. Ele pode ser usado com qualquer provider, como veremos neste artigo.

Além desses componentes, temos ainda o fbCommandBuilder, que gera automaticamente comandos SQL para efetivar no banco de dados as mudanças feitas em um DataSet. O namespace que você deve utilizar para acessar todos os tipos do provider é FirebirdSql.Data.Firebird.

O provider para Firebird tem ainda algumas exclusividades, se comparado aos demais providers para ADO.NET. Temos no namespace FirebirdSql.Data.Firebird.Services, classes e objetos que permitem utilizar vários serviços e recursos do servidor Firebird, como: Backup, Restore, acesso a logs do servidor, manipulação de contas de usuário e configurações de segurança, recuperação de dados estatísticos do servidor etc. Consulte a documentação do SDK que acompanha o provider para saber mais sobre esses recursos.

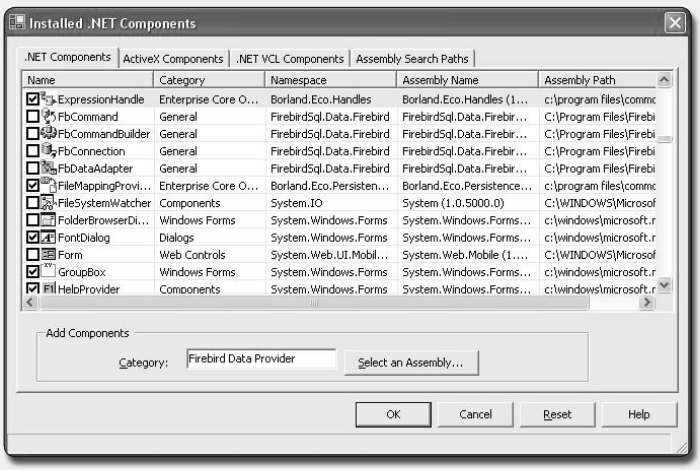


Figura 10. Instalação do provider .NET no Delphi

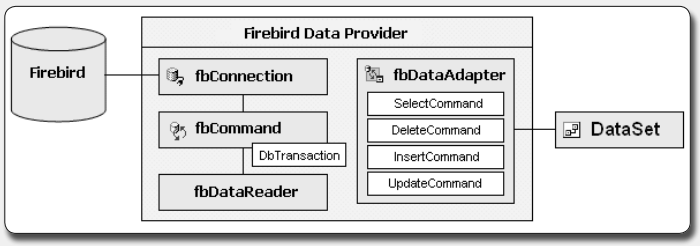


Figura 12. Arquitetura de componentes do Firebird Data Provider

Figura 11. Componentes do provider .NET Firebird já instalados

Listagem 3. Codificando o cadastro de dados com Stored Procedures

```
procedure TfrmCustomer.btnCadastrarClick(Sender: TObject);
begin
  try
    with SQLStoredProc1 do
    begin
      ParamByName('CLIENTE').AsString := edtCliente.Text;
      ParamByName('CONTACT_FIRST').AsString := edtContactFirst.Text;
      ParamByName('CONTACT_LAST').AsString := edtContactLast.Text;
      ParamByName('PHONE_NO').AsString := edtFone.Text;
      ParamByName('ADDRESS_LINE1').AsString := edtEndereco.Text;
      ParamByName('PAIS').AsString := edtPais.Text;
      ExecProc;
      ShowMessage('O registro foi inserido com o código ' +
        IntToStr(ParamByName('CODIGO').AsInteger));
    end;
  except on E:Exception do
    ShowMessage('Ocorreu o seguinte erro : E.Message);
  end;
end;
```

Listagem 4. Código do evento Click para cadastro de cliente

```
interface
…
const
  strConexao = 'User=SYSDBA;Password=masterkey;
  Database=<Caminho>LOCADORA.FDB;';
…

procedure TWebForm1.btnCadastrar_Click(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  Comand : FbCommand;
  pCliente : FbParameter;
  pContato1 : FbParameter;
  pContato2 : FbParameter;
  pTelefone : FbParameter;
  pEndereco : FbParameter;
  pPais : FbParameter;
begin
  try

    { Criação dos objetos de conexão }
    Conn := FbConnection.Create;

    { Atribuição da string de conexão e abertura do BD}
    Conn.ConnectionString := strConexao;
    Conn.Open;

    { Atribuição e chamada do procedimento }
    Comand := FbCommand.Create('select codigo from ins_customer(?, ?, ?, ?, ?, ?)',Conn);

    { Limpeza, Criação e Atribuição dos parâmetros }
    Comand.Parameters.Clear;
    pCliente := FbParameter.Create;
    Comand.Parameters.Add(pCliente);
    Comand.Parameters[0].Value := txtCliente.Text;
    pContato1 := FbParameter.Create;
    Comand.Parameters.Add(pContato1);
    Comand.Parameters[1].Value := txtContato1.Text;
    pContato2 := FbParameter.Create;
    Comand.Parameters.Add(pContato2);
    Comand.Parameters[2].Value := txtContato2.Text;
    pTelefone := FbParameter.Create;
    Comand.Parameters.Add(pTelefone);
    Comand.Parameters[3].Value := txtTelefone.Text;
    pEndereco := FbParameter.Create;
    Comand.Parameters.Add(pEndereco);
    Comand.Parameters[4].Value := txtEndereco.Text;
    pPais := FbParameter.Create;
    Comand.Parameters.Add(pPais);
    Comand.Parameters[5].Value := txtPais.Text;
    Comand.ExecuteNonQuery;
    Response.Write('Cliente cadastrado com sucesso!');
  finally
    comand.free;
  end;
end;
```

A primeira coisa que podemos ver na **Listagem 4** é que criamos uma constante na área *Interface* da página informando a *string* de conexão com o banco. Esse valor será usado para que o componente *fbConnection* acesse o BD. Já no evento de *Click*, criamos dois componentes em *run-time* que são:

• *Conn*: Um *fbConnection* para conexão direta com o *BD*;

• *Command*: Componente *fbCommand* para execução de comandos;

Além dos dois componentes, criamos também os parâmetros que são necessário para a inserção do registro. Logo de início, criamos todos os componentes e atribuímos a *string* de conexão com o banco ao *Conn*, componente de conexão. Em seguida criamos o objeto para consultas na base de dados e atribuímos a ele a chamada da *Stored Procedure*, na qual temos nossos parâmetros de entrada. A configuração dos parâmetros ocorre logo em seguida, quando, individualmente, todos são criados e recebem um valor.

Depois de atribuído valor a todos os parâmetros, chamamos o método *ExecuteNonQuery* para que nosso procedimento seja executado e exibimos uma mensagem de retorno ao usuário. Finalizadas estas configurações, execute o projeto e faça o cadastro de um novo cliente.

Conclusão

Conhecemos neste artigo os conceitos básicos e referências de uso para *Stored Procedures*, adquirindo assim uma noção básica e simples exemplos de como utilizar esse recurso que deverá ser mais explorado em nossos sistemas. Com auxílio da ferramenta do IBExpert, veja como criar outras situações de procedimentos, utilizar condições de cadastros e verificação de registros. Abraço e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Consumindo Web Services

Aprenda como utilizar na prática recursos de um Web Services e como tirar proveito deles

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da Equipe Editorial ClubeDelphi.

Edinei Daniel Steffen
(edineidaniel@gmail.com)
é técnico e Bacharelando em Sistemas de Informação, Analista e Desenvolvedor de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL.

Com o advento da Internet e seu rápido avanço tanto na parte tecnológica quanto em serviços, cada vez mais existe a necessidade de compartilhamento de informações. Serviços estes que podem ser responsáveis por um simples cálculo, uma simples consulta ou até mesmo uma avançada transação de produtos entre empresas, necessitando apenas de um Web Service, ou seja, uma aplicação que funcione com a utilização de um serviço público na internet ou na rede local.

Lembrando que Web Service não traz consigo o conceito de aplicações com interfaces gráficas para usuários, mas sim serviços que são acessados e disponibilizados em uma linguagem universal XML("Extensible Markup Language"), ou linguagem de marcação, possibilitando com isso uma comunicação padronizada em um único modelo (XML). Comunicação esta que normalmente é feita através do protocolo HTTP ("Hypertext Transfer Protocol" – "Protocolo de Transferência de Hipertexto"), fazendo com que sistemas de diferentes linguagens e plataformas troquem dados entre si. Um termo bastante comentado no mundo dos analistas, desenvolvedores e arquitetos de software, mas pouco posto em prática devido à falta de entendimento da prática de utilização do mesmo.

Veremos neste artigo que a implementação desta tecnologia utilizando o Delphi 7. Para tanto faremos o uso e descrição dos serviços de envio de e-mails, conversor de temperaturas e o serviço de consulta de CEP e cidades e a partir destes desenvolveremos as conexões e posteriormente faremos a utilização dos mesmos.

Nota: A criação e utilização de Web Services está disponível no Delphi a partir da versão 6.0 e hoje encontramos em todas as versões disponíveis incluindo RAD Studio 2007.

Nota: O escopo deste artigo tem como finalidade mostrar apenas como consumir, ou seja, utilizar Web Services já disponíveis no mercado. Está fora do escopo mostrar como criar WS.

Desmistificando a tecnologia

Devido a essa tecnologia trabalhar com a transferência de dados através do protocolo HTTP, não existe motivo algum para pensar que esses dados estarão expostos na Internet, pois os mesmos permanecem encapsulados pelo protocolo SOAP ("Simple Object Access Protocol"), fazendo com que assim exista um isolamento entre o serviço e o banco de dados. Assim sendo, para existir uma comunicação com qualquer WebService é necessário uma implementação do protocolo SOAP determinado no W3C (órgão que desenvolve padrões para a criação e a interpretação dos conteúdos para a Web).

Antes de partirmos para a prática em si, necessitamos de alguns esclarecimentos sobre os termos que serão envolvidos durante a utilização de um Web Services:

• *WSDL ("Web Services Description Language"),* definida como sendo a linguagem de um WebService, baseado em XML, contendo os métodos, operações e o modo de como acessá-lo;

• *UDDI ("Universal Description, Discovery and Integration"),* é uma especificação que define um serviço de Web Services, nesta contém informações sobre provedores, implementações e serviços;

Analogia para o Uso

Vamos pensar de uma forma diferente para representar a utilização de um WebService.

• Imagine-se necessitando de dinheiro;

• Você procura na sua cidade ("UDDI") o melhor banco;

• No banco você escolhe uma linha de crédito ("WSDL");

• Em seguida, você faz o pedido ao atendente ("SOAP"), que leva o pedido até a gerência ("HTTP");

• O gerente prepara a análise de crédito para sua escolha ("Classe que implementa o serviço");

• Depois de pronta, o atendente traz a papelada para você usufruir do dinheiro ("SOAP de resposta");

• É, basicamente, assim que funcionam os Web Services.

Fazendo o uso dos Serviços

Iremos utilizar para este artigo a versão do Delphi 7 em uma aplicação Win32, mas fique à vontade para o uso de qualquer outra versão superior para dar início à implementação Web Services em seus sistemas.

Exemplo 01: Envio de E-mails

No menu *File|New>Application* crie uma nova aplicação, altere a propriedade *Name* do formulário inicial para "frmWSEmail" e a propriedade *Caption* para "Web Service 01: Envio de Email". Salve a *Unit* principal do projeto como "uWSEmail.pas" e o projeto salve como "prjWSEmail.dpr".

Em seguida vamos à montagem do layout do formulário, adicione um *Label* no formulário com sua propriedade *Caption* para "Nome do Remetente". Ao lado do *Label* adicione um *Edit (*"edtNomeRem"*)* e limpe a propriedade *Text*. Abaixo adicione uma nova *Label* e em sua propriedade *Caption* configure para "E-mail do Remetente" e ao lado acrescente um *Edit*("edtEmailRem"). Após isso adicione um *Label* no formulário com sua propriedade *Caption* igual a "Nome do Destinatário". Da mesma forma ao lado da *Label* adicione um *Edit* ("edtNomeDes") e limpe a propriedade *Text*. Abaixo adicione uma nova *Label* e em sua propriedade *Caption* altere para "E-mail do Destinatário" acrescentando ao seu lado um *Edit*("edtEmailDes"). Necessitando ainda um campo onde o usuário informa o assunto seguindo o mesmo modelo dos demais campos utilizando um *Label* e um *Edit*("edtAssunto"). E por fim, utilizaremos um componente *Memo* onde será possível o usuário digitar a mensagem do e-mail, portanto adicione um componente *Memo*("memMensagem"). Adicione também um botão que será responsável por acionar o Web Services (**Figura 1**).

Agora deve ser importado no projeto o *WSDL*("Web Services") que disponibiliza o serviço de envio de e-mail,

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como trabalhar Web Services e SOAP.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=6733&hl=

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como trabalhar Web Services e SOAP.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=6771&hl=

Nota do DevMan

Segundo o verbete pt.wikipedia.org/wiki/Soap: SOAP (originado do acrônimo inglês Simple Object Access Protocol) é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída, utilizando tecnologias baseadas em XML. Sua especificação define um framework que provê maneiras para se construir mensagens que podem trafegar através de diversos protocolos e que foi especificado de forma a ser independente de qualquer modelo de programação ou outra implementação específica. Por não se tratar de um protocolo de acesso a objetos, o acrônimo não é mais utilizado.

Geralmente servidores SOAP são implementados utilizando-se servidores HTTP, embora isto não seja uma restrição para funcionamento do protocolo. As mensagens SOAP são documentos XML que aderem a uma especificação fornecida pelo órgão W3C ("Órgão que regulamenta e cria padrões de internet").

O primeiro esforço do desenvolvimento do SOAP foi implementar RPC's sobre XML.

Nota: A Chamada de procedimento remoto ou RPC ("Remote Procedure Call") é o tipo de protocolo para chamada remota de procedimentos em qualquer lugar da rede ou uma chamada de função para o método de transferência de controle de parte de um processo para outro, permite a divisão de um software em várias partes, compartilhamento de arquivos e diretórios. O protocolo RPC pode ser implementado sobre diferentes protocolos de transporte, o RPC não especifica como a mensagem é enviada, somente especifica e interpreta. As camadas de transporte que o RPC implementa são a TCP e a UDP.

RPC e Sockets têm a função de implementar numa aplicação as funções de rede, mas fazem isto de maneiras diferentes.

Definição
Envelope das mensagens, regras de codificação, convenção RPC, ligação com protocolos subjacentes.

O SOAP oferece

* Mecanismo para definir a unidade de comunicação;

* Mecanismo para lidar com erros;

* Mecanismo de extensão que permite evolução;

* Mecanismo entre as mensagens SOAP e o HTTP, representar tipos de dados em XML;

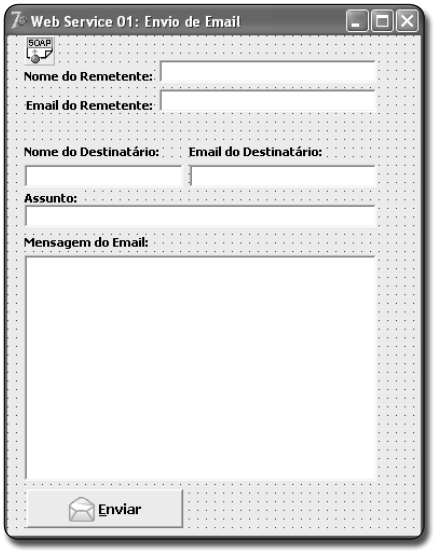


Figura 1. Layout do Formulário

o link deste serviço é *www.abysal.com/soap/AbysalEmail.wsdl*. Se digitar esse endereço no *browser*, notará que ele lhe retornará uma página semelhante à **Figura 2**. Esse retorno nada mais é do que o arquivo XML com as configurações do Web Services.

A importação do *WSDL* no Delphi é extremamente simples, basta ir até o menu *File\New>Other>Web Services>WSDL Importer* (**Figura 3**).

Ao confirmar a caixa de diálogo, o Delphi abrirá outra janela onde é preciso informar o link do *WSDL* (**Figura 4**). Na última tela do assistente já teremos uma pré-visualização dos métodos que compõem a interface do Web Services. Clicando no botão *Finish* o Delphi criará uma *Unit* com os métodos de acesso e interação com o serviço, e neste momento já é possível salvar, na mesma pasta do projeto, esta nova *unit*, com o nome sugerido pelo Delphi.

Basta agora utilizar o serviço, mas para isso necessitamos de um componente "*HTTPRIO*", que é o componente responsável por transformar chamadas a métodos de interface, escritas em Pascal, em requisições SOAP. Em seguida, configurar sua propriedade *Url*, com o endereço do serviço, conforme ilustrado na **Figura 4.** Por isso inclua o compo-

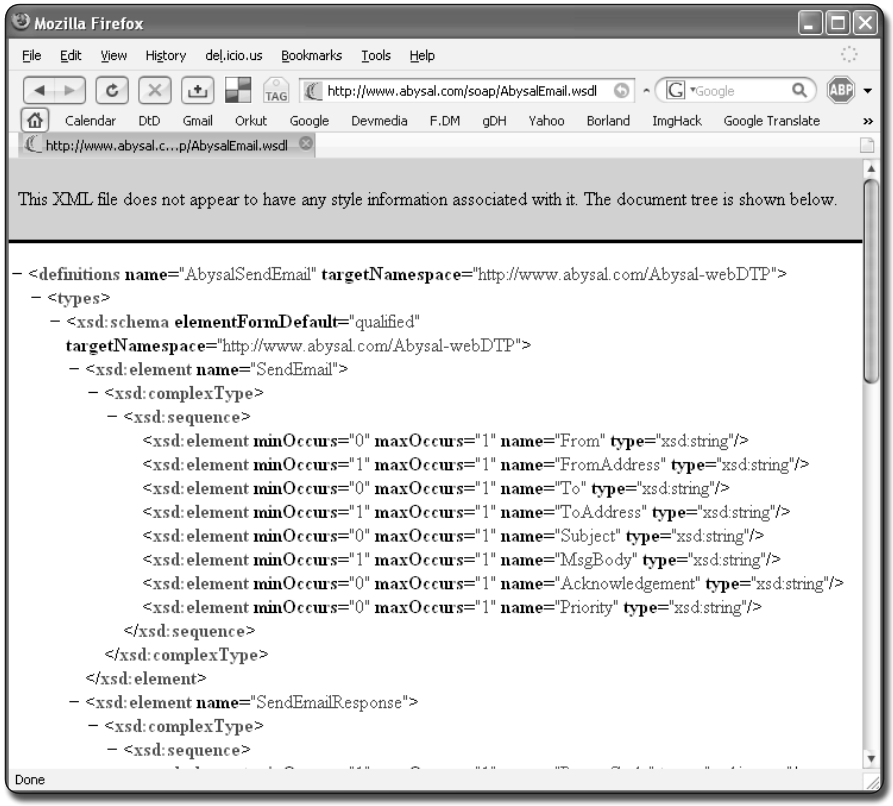


Figura 2. WSDL do Web Service

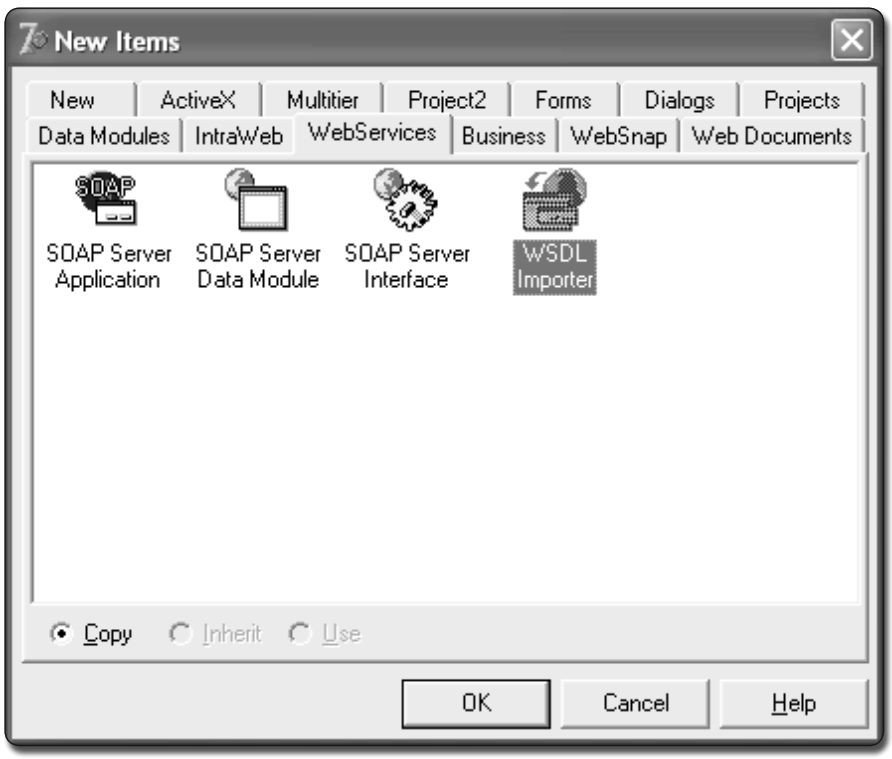


Figura 3. Opção do WSDL Importer

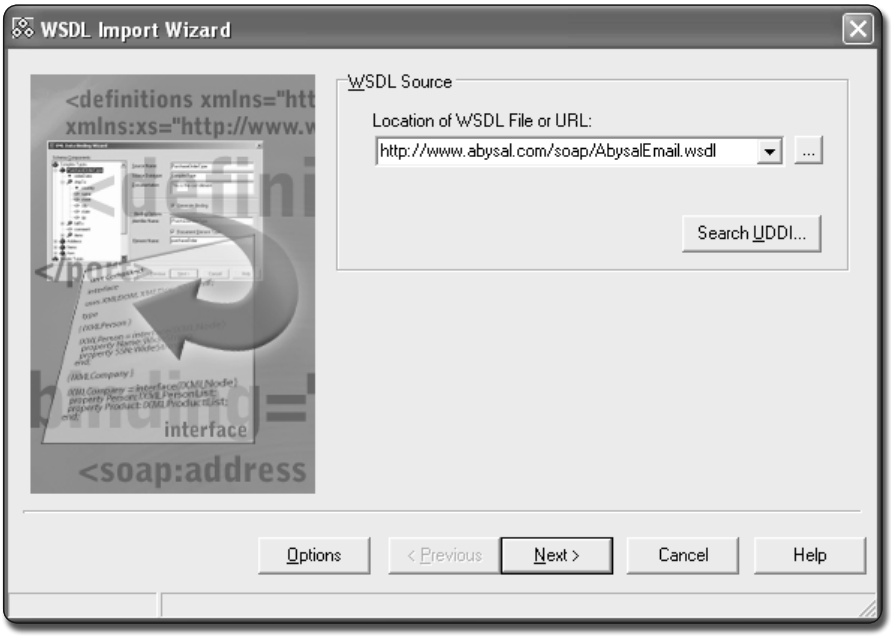


Figura 4. Assistente de importação do WSDL

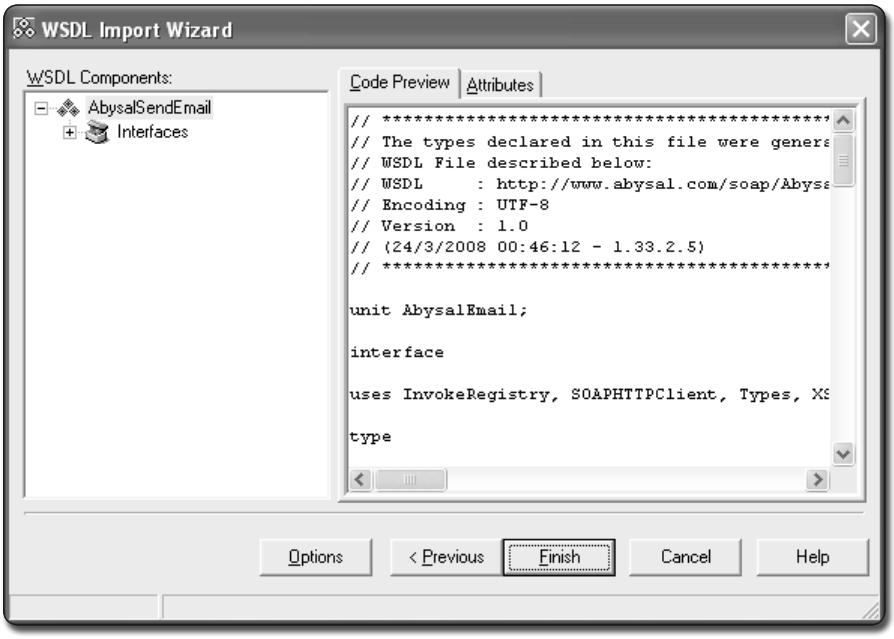


Figura 5. Unit gerada

Listagem 1. Envio de e-mail utilizando Web Service

```
procedure TfrmWSEmail.bbtnEnviarClick(Sender:
  TObject);
begin
  try
    (HTTPRIO1 as SendEmailPortType).SendEmail
      (edtNomeRem.Text, edtEmailRem.Text,
       edtNomeDes.Text,  edtEmailDes.Text,
       edtAssunto.Text, memMensagem.Lines.Text,
       '', '');
    ShowMessage('E-mail enviado com Sucesso!');
  except
    ShowMessage('Problemas na conexão, com o
      serviço de envio de E-mail');
  end;
end;
```

nente indicado e digite o link do Web Service na propriedade mencionada anteriormente. O componente pode ser encontrado na paleta de componentes *WebServices*.

Após a configuração do componente "*HTTPRIO*", vamos relacionar a *Unit* do formulário com a *Unit* criada há pouco pelo Delphi (**Figura 5**), pressionando as teclas *ALT + F11* e selecionar a *Unit* que contém os métodos. Agora podemos passar para a parte de codificação e utilização dos métodos que são disponibilizados pelo Web Service.

A codificação se dará no evento *OnClick* do botão "Enviar" onde deverá digitar o código da **Listagem 1,** que fará a chamada ao serviço da Web importado, passando os valores requisitados para a função encapsulada no serviço, enviando assim o e-mail e retornando uma mensagem ao usuário (**Figura 6**).

> **Nota**: A palavra "SendEmailPortType", apresentada no código da **Listagem 1** diz respeito ao nome da interface do serviço e não ao nome da *Unit* ao qual os métodos foram disponibilizados.

Exemplo 02: Transformação de Temperaturas

Este exemplo tratará da implementação de um Web Service bem simples, cujo objeto é converter uma temperatura dada

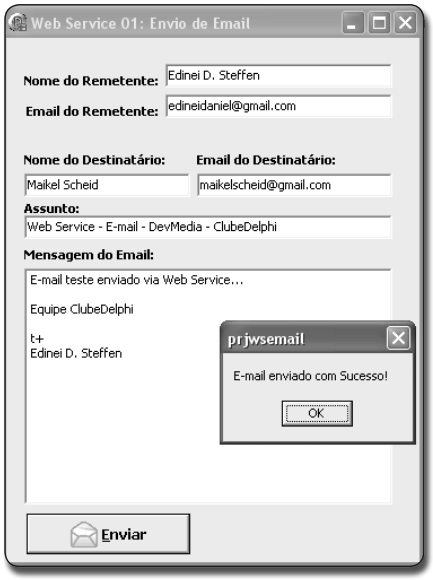


Figura 6. Resultado da implementação do Web Service no Projeto

em graus *Celsius* para uma temperatura na escala *Fahrenheit* e vice-versa.

Iniciando um novo projeto, no menu *File|New>Application* crie uma nova aplicação, altere a propriedade *Name* do formulário inicial para "frmWSConverteTemp" e a propriedade *Caption* para "Web Service 02: Conversão de Temparaturas (CtoF, FtoC)". Salve a *unit* principal do projeto como "uWSConverteTemp.pas" e o projeto salve como "prjWSConverteTemp.dpr".

Para o layout do formulário, utilizaremos primeiramente um componente *RadioGroup*("rgConfigTemp"), que será responsável pela configuração do tipo de conversão que se deseja realizar. Na propriedade *Caption* será descrito com "Tipo de Conversões", na propriedade *Items* serão adicionadas duas linhas. A primeira com a frase "Celsius para Fahrenheit" e na segunda linha "Fahrenheit para Celsius". E para uma das alternativas ficar marcada será informado o número 0 (zero) na propriedade *ItemIndex*.

Ainda será necessário um *Label* com sua propriedade *Caption* alterado para "Informe a Temperatura", logo abaixo da *Label* um componente *Edit*("edtTemp"), para que o usuário possa informar o valor da temperatura a ser convertida e limpe a propriedade *Text*. Para mostrar o resultado da conversão será utilizado um componente *Label* ("lblResultado") e para finalizar a parte de layout será adicionado um *Button* que será responsável por executar os comandos de interação com Web Service (**Figura 7**).

Repita o processo que realizamos no exemplo anterior para importar o WSDL de conversão, ou seja, acessando o menu *File|New>Other>Web Services>WSDL Importer* e digitando o endereço *developerdays.com/cgi-bin/tempconverter.exe/wsdl/ITempConverter*.

Feita a importação, inclua um componente "*HTTPRIO*" e informe a *Url* do serviço. Não devemos esquecer de salvar a *Unit* criada pelo delphi a partir da importação do *WSDL*. O nome da *Unit* pode seguir com o nome que o Delphi sugere.

No evento *OnClick* do botão "Buscar", faremos o uso do código da **Listagem 2**, sem antes esquecer de relacionar as *Unit's* (ALT + F11). Após o código implementado obtém-se o resultado, conforme ilustrado na **Figura 8.**

Listagem 2. Código de utilização do Web Services para conversão de temperatura

```
case rgConfigTemp.ItemIndex of
  0 : lblResultado.Caption :=
    IntToStr((HTTPRIO1 as ITempConverter ).
      CtoF(StrToInt(edtTemp.Text)));
  1 : lblResultado.Caption :=
    IntToStr((HTTPRIO1 as ITempConverter ).
      FtoC(StrToInt(edtTemp.Text)));
end;
```

Listagem 3. Implementação da mudança de formulário de acordo com a pesquisa

```
procedure TfrmWSCEP.rgTipoConsultaClick(Sender:
  TObject);
begin
  case rgTipoConsulta.ItemIndex of
    0:
      begin
        { Mostra o componente Panel com os campos
          na tela }
        pnlParametros.Visible := False;
        { Monta a descrição do campo principal de
          consulta }
        lblCepDescricao.Caption := 'Informe o CEP';
    end;
    1:
      begin
        { Oculta o componente Panel com os campos na tela }
        pnlParametros.Visible := True;
        { Monta a descrição do campo principal de consulta }
        lblCepDescricao.Caption := 'Informe a parte da
          descrição do Lograd.:';
    end;
  end;
end;
```

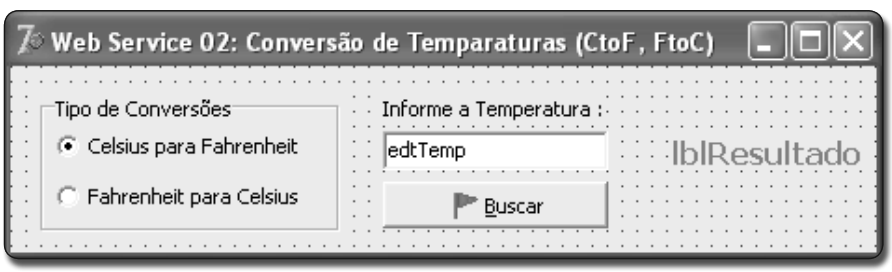


Figura 7. Layout do Formulário.

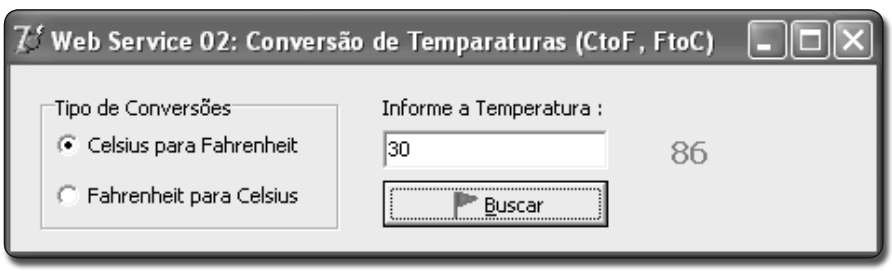


Figura 8. Resultado Final, após a implementação do Web Service

Exemplo 03: Consulta de CEP e Logradouros

Este exemplo tratará da implementação de um Web Services, cujo objetivo é retornar uma descrição completa de um ou vários logradouros, dependendo da forma que vão ser informados os parâmetros para serviço. Este Web Service foi encontrado na Internet, portanto não sei informar, por quanto tempo os dados estarão disponíveis, mas segundo a documentação, este serviço dispõe de praticamente todas as cidades/logradouros do país. O serviço disponibiliza dois métodos para

realizar a consulta, um onde somente é necessário informar o CEP retornando o Logradouro, Cidade e UF e outro onde é possível realizar um busca aproximada pela descrição do logradouro, retornando todos os registros que se encaixam neste tipo de filtro.

Novamente iniciaremos um novo projeto no menu *File|New>Application* e altere a propriedade *Name* do formulário inicial para "frmWSCep" e a propriedade *Caption* para "Web Service 03: Consulta de CEP e Cidades". Salve a *unit* principal do projeto como "uWSCep.pas" e o projeto salve como "prjWSCep.dpr".

Vamos para a montagem do layout do formulário que irá interagir com o Web Service. Colocaremos os seguintes componentes em tela, um *RadioGoup*("rgTipoConsulta") e na sua propriedade *Items* adicione dois itens ("Por CEP", "Por Descrição de Logradouro"). Abaixo deste componente, adicionaremos uma *Label*("lblCepDescricao") com o *Caption* "Informe o CEP". Adicionaremos abaixo da *Label* um componente *Edit* ("edtCepDescricao"). Utilizaremos uma *Panel*("pnlParametros"), sua propriedade *Visible* ficará como "False" e dentro desta *Panel* será adicionado mais quatro componentes, dois *Edit*`s, um sendo identificado por "edtUF" e ao seu lado uma *Label* com a descrição do campo, e o outro *Edit*("edtNroReg"), com uma *Label* ao seu lado para identificação. Será adicionado um *BitBtn*("bbtnBuscar") e por fim um componente *Memo*("memResultado") levando também ao seu lado uma *Label* para descrever a finalidade do campo conforme layout apresentado na **Figura 9.**

O componente *Panel* será exibido conforme a opção de consulta selecionada. No evento *OnClick* do "rgTipoConsulta" implemente o código da **Listagem 3** que se encontra comentado de acordo com a ação a ser codificada.

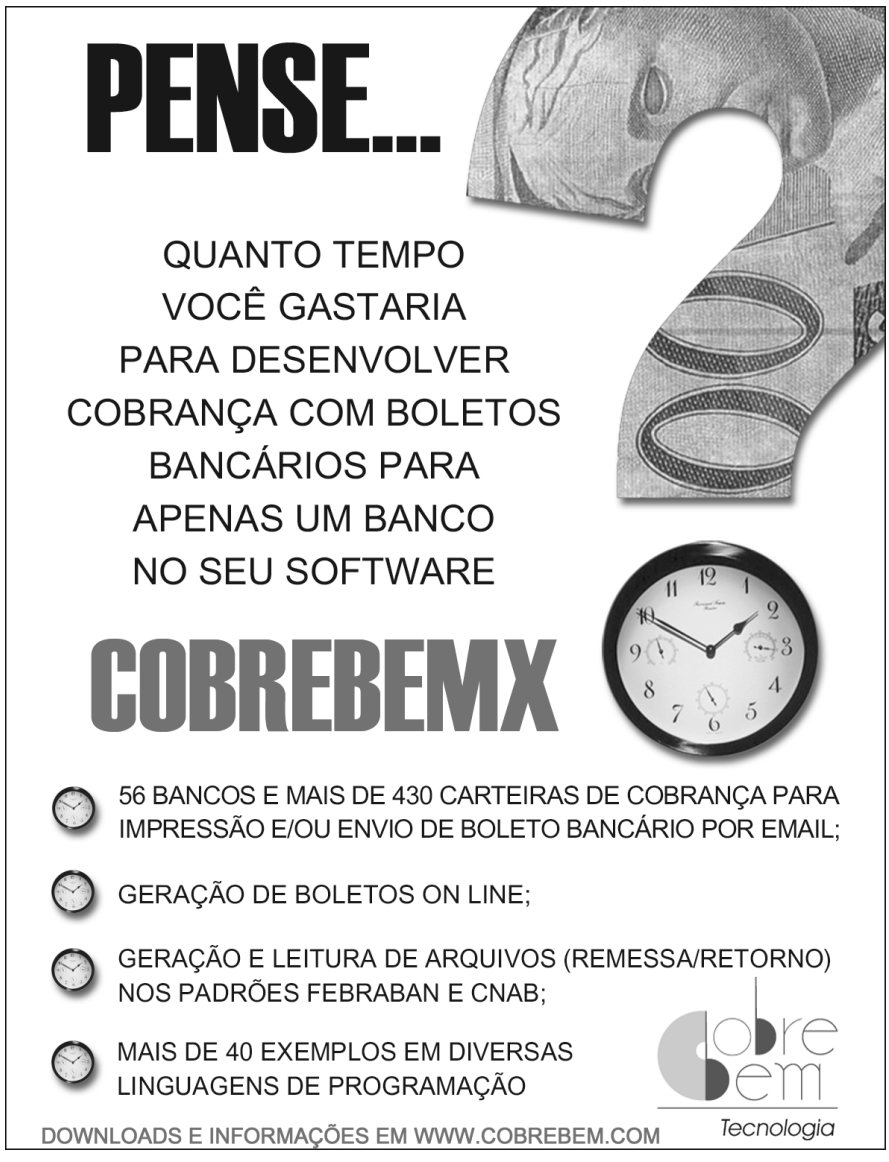
Em seguida podemos realizar a importação do *WSDL* do serviço para nossa aplicação, seguindo os mesmos passos dos dois exemplos anteriores. Dessa vez basta usar a *url desenvolvimento.setrem.com.br/wsCEP/wsCEP.exe/wsdl/ICEP*. No menu *File|New>Other>Web Services>WSDL Importer* informe o link mencionado referente ao serviço, seguindo os passos até finalizar o assistente. Após a importação do serviço salve a *Unit* na mesma pasta do projeto.

Agora vamos adicionar um componente *HTTPRIO* para realizar a interação com os métodos do serviço, e em sua propriedade URL inclua o link indicado.

Nota: De onde vem o link para preencher a propriedade *Url* do componente *HTTPRIO*? Pois bem, a *Unit* gerada a partir do *WSDL Importer* existe uma única função implementada que é responsável por realizar a comunicação com o Web Services. E é nessa função que existe uma variável identificando a *Url* que o componente necessita, nesse caso: *desenvolvimento.setrem.com.br/weCEP/wecCEP.exe/soap/ICEP*.

Após feita esta configuração o Web Service está pronto para ser consumido, portanto, necessitamos relacionar as *Unit`s* (ALT + F11) e no evento *OnClick* do botão "Buscar" adicionaremos o código da **Listagem 4**.

Como podemos ver, o código segue os mesmos padrões de codificação dos exemplos anteriores. Passamos para o WS os parâmetros necessários e ele nos retorna o CEP encontrado. O resultado é demonstrado através das **Figuras 10** e **11**.



Listagem 4. Código de requisição do CEP

```
case rgTipoConsulta.ItemIndex of
  0: memResultado.Lines.Text :=
    (HTTPRIO1 as ICEP).LogradourosPorCep(
      edtCepDescricao.Text);
  1: memResultado.Lines.Text := (HTTPRIO1 as ICEP).
    LogradourosPorNome(edtCepDescricao.Text,
    edtUF.Text, edtNroReg.Text);
end;
```

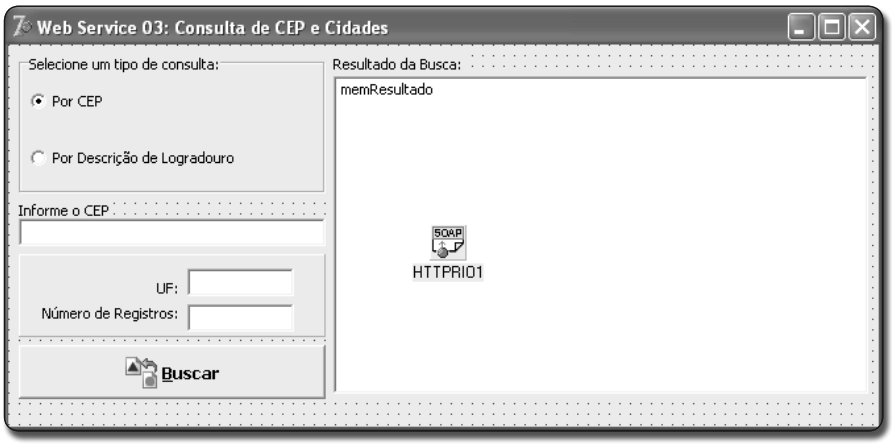


Figura 9. Layout do Formulário, após o desenho do mesmo

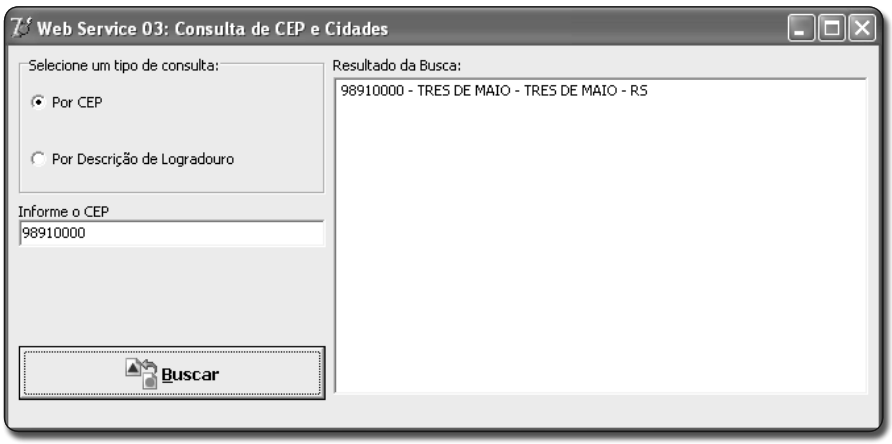


Figura 10. Demonstração da Consulta por CEP

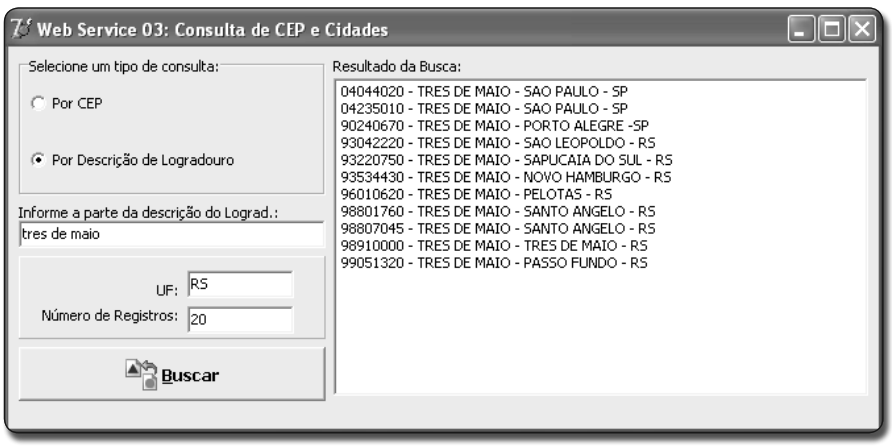


Figura 11. Demonstração da Consulta por Descrição do Logradouro

Da mesma forma é demonstrado o resultado, se utilizada a opção "Por Descrição de Logradouro", conforme ilustrado na **Figura 11.**

Com estes exemplos podemos ter uma boa noção dos recursos que a utilização de Web Services nos oferece. Inúmeras são as possibilidades de implementações em termos de desenvolvimento de software.

Conclusão

Foram abordados neste artigo os principais conceitos sobre a Web Services, juntamente com exemplos práticos sobre a utilização destes serviços, demonstrando a facilidade de se trabalhar com esse tipo de tecnologia.

Utilizamos padrões abertos como HTTP, SOAP, UDDI, ao invés de tecnologias proprietárias, e tudo isso ainda baseado em um padrão universal de comunicação entre sistemas, o XML.

Agora cabe a você, analisar o emprego desta tecnologia em seu cotidiano, e aplicar, com base nas funcionalidades apresentadas neste artigo, mais recursos em suas aplicações. Explore mais esta área, busque mais, leia mais, aprenda mais, pois o emaranhado de opções que existem nesse tipo de serviço é muito grande. Em uma próxima oportunidade veremos aqui na Easy Delphi como implementar aplicações servidoras.

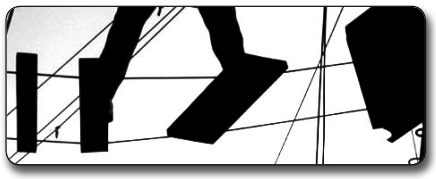
Um grande abraço e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura pode estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

DevMedia GROUP

Para mais informações:
www.devmedia.com.br/central

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Orientação a Objetos no Delphi for PHP

Como aplicar conceitos de POO em aplicações PHP – Parte 3

Rodrigo Carreiro Mourão
(rodrigocarreiro@tdstecnologia.com.br)
é consultor da TDS Tecnologia – RJ atuando na área de desenvolvimento de projetos Orientados a Objetos, Design Patterns, MVC. BDS2006 Win32 Product Certified. Instrutor de treinamentos oficiais Delphi for Win32, Delphi for PHP, Delphi .Net. Palestrante do BorCon 2007. É mantenedor do site www.delphisophp.com desenvolvido totalmente em Delphi for PHP.

Olá, ainda nesta linha de OOP no Delphi for PHP neste terceiro e último artigo trataremos de um assunto pouco conhecido pela maioria dos programadores, mas que é fundamental para um bom desenvolvimento: os padrões de projeto. Como podemos observar, a orientação a objetos facilita e muito a nossa vida em relação aos códigos que escrevemos e a manutenção desses códigos, porém conforme vamos avançando em nossos projetos vamos nos deparando com situações delicadas que nos tomam um tempo precioso. Problemas recorrentes que se repetem em cada novo projeto que iniciamos. Se reparar bem todos os programadores têm aquela famosa *Unit* "Faz Tudo" onde temos rotinas prontas que nos ajudam a resolver estes problemas corriqueiros.

Mas em projetos grandes onde temos um a equipe envolvida esta *Unit* não ajudará muito. É preciso que haja uma linguagem universal que todos entendam, ou melhor, é preciso que se desenvolva dentro de um padrão, mas o grande problema e que cada equipe pode ter o seu padrão, cada empresa sua linha de raciocínio e aquela velha máxima "Cada um na sua, mas com alguma coisa em comum".

Enfim, um ponto final foi colocado neste dilema em 1995 com o surgimento dos Design Patterns ou Padrões de Projeto para Sistemas Orientados a Objetos.

Histórico

O conceito de padrão de projeto foi usado pela primeira vez na década de 70 pelo arquiteto e urbanista Austríaco Cristopher Alexander. Ele observou na época que as construções, embora fossem diferentes em vários aspectos, todas passavam pelos mesmos problemas na hora de se construir. Eram problemas que se repetiam em todas elas e na maioria das vezes numa mesma fase da construção.

Foi aí que Cristopher resolveu documentar esses problemas e mais do que isso, passou também a documentar as soluções que eram aplicadas para resolução destes problemas. Entenda que até agora não há nada de programação ou informática e sim projetos, plantas, métricas, definições. Neste momento surgiam os padrões de projetos para a engenharia civil. Padrões esses que descreviam os problemas recorrentes em um projeto de engenharia e a solução reutilizável para este problema. Em seus livros *Notes on the Synthesis of Form, The Timeless Way of Building* e *A Pattern Language,* Cristopher estabelece que um padrão deve Ter as seguintes características:

- Encapsulamento;
- Generalidade;
- Equilíbrio;
- Abstração;
- Abertura;
- Combinatoriedade;

Ou seja, um padrão deve ser independente, deve permitir a construção de outras realizações, deve representar abstrações do conhecimento cotidiano, deve permitir a sua extensão para níveis mais baixos de detalhe e deve ser relacionado hierarquicamente.

Além destas características, Alexander definiu que um padrão deve ser descrito em 5 partes:

- Nome;
- Exemplo;
- Contexto;
- Problema;
- Solução;

Padrões de Projetos - Design Patterns

Anos mais tarde Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides iniciaram suas pesquisas baseadas nos trabalhos de Alexander. Também conhecidos como GOF ("Gang of Four") eles começaram a descrever e documentar os problemas e soluções para desenvolvimento de softwares orientados a objetos e em 1995 lançaram o livro que ser tornou um fenômeno na área de análise e desenvolvimento de software: *Design Patterns: Elements of Reusable Object-Oriented Software.*

Neste momento iniciava-se uma nova fase no desenvolvimento de sistemas, pois agora havia um padrão a ser seguido, e cada padrão apresentando uma solução que poderia se reutilizada várias vezes para solucionar aqueles problemas recorrentes no desenvolvimento de software. Os padrões GOF, como são conhecidos, se dividem em três grandes categorias: *criacionais, estruturais* e *comportamentais,* somando 23 no total.

Padrões criacionais

- Abstract Factory;
- Builder;
- Factory Method;
- Prototype;
- Singleton;

Padrões estruturais

- Adapter;
- Bridge;
- Composite;
- Decorator;
- Façade;
- Flyweight;
- Proxy;

Padrões comportamentais

- Chain of Responsability;
- Command;
- Interpreter;
- Iterator;
- Mediator;
- Memento;
- Observer;
- State;
- Strategy;
- Template Method;
- Visitor;

Nas **Figuras 1, 2 , 3, 4** e **5** temos os diagramas UML de alguns Padrões de Projeto descritos acima.

Interfaces

Falar de Padrões de Projetos sem falar de *interfaces* é como cometer um crime. A maioria dos padrões utilizam *interfaces* para serem implementados. Mas o que vem a ser uma *interface*?

Em OO temos a herança como recurso para reaproveitar códigos e facilitar a manutenção, mas a herança

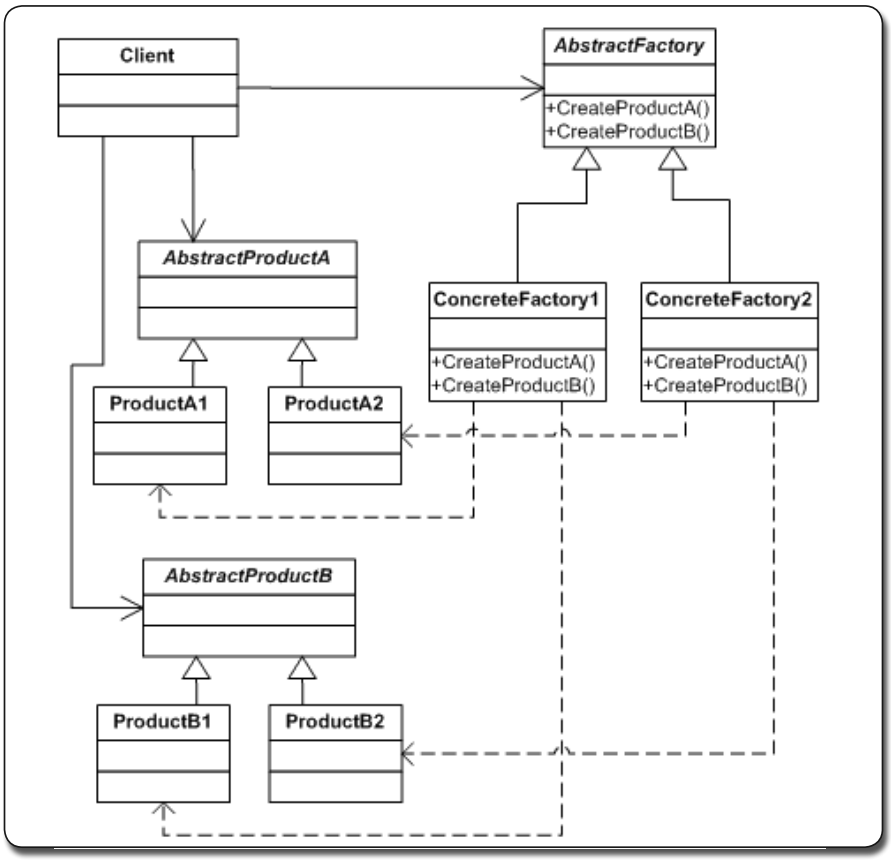


Figura 1. Diagrama UML para padrão Abstract Factory

Listagem 1. Utilizando Interfaces

```
<?php
  interface IDAO{
    function Gravar();
  }
  class ClienteDAO implements IDAO{
    function Gravar(){
      echo "Gravando Cliente !!";
    }
  }
  class FornecedorDAO implements IDAO{
    function Gravar(){
      echo "Gravando Fornecedor !!";
    }
  }
  function GravarObjeto($Obj){
    if ($Obj instanceof IDAO){
      $Obj->Gravar();
    }else{
      echo "O objeto passado nao suporta a interface IDAO";
    }
  }
?>
```

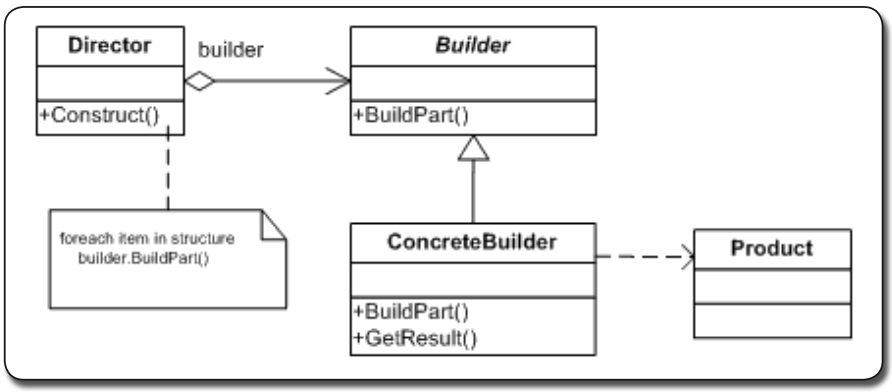


Figura 2. Diagrama UML para padrão Builder

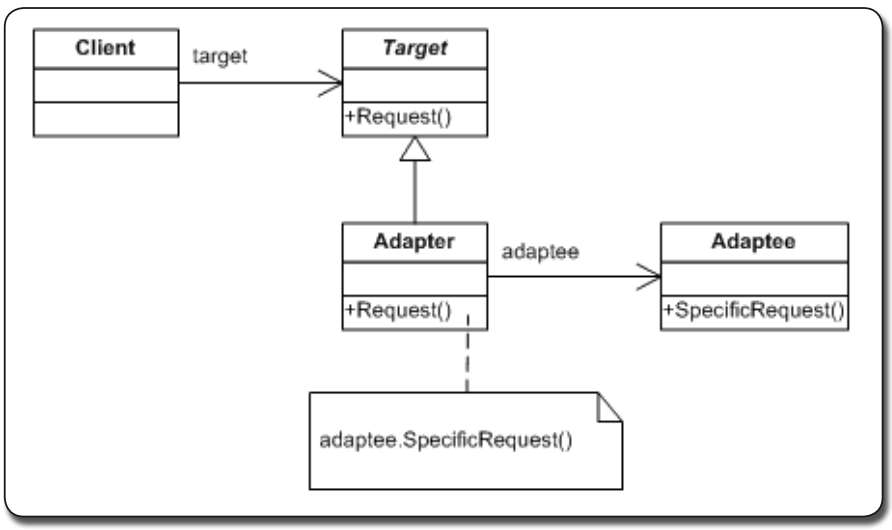


Figura 3. Diagrama UML para padrão Adapter

em algumas linguagens como o PHP restringem-se a uma única classe. Não podemos herdar de duas ou mais classes ao mesmo tempo. Há linguagens como o C que permite o uso de herança múltipla, porém os desenvolvedores do PHP optaram pelo uso de *interfaces*.

Interfaces nada mais são do que uma espécie de regra que a classe que a implementa deverá seguir ou ter em sua estrutura. Voltando àquela analogia básica que sempre usamos em OO, uma classe é como a planta de uma casa, algo ainda intangível, no papel. E o objeto é a casa construída, algo concreto. Assim podemos aqui colocar a *interface* como um contrato de licitação onde estarão todas as cláusulas que ditam as regras e normas para a construção desta casa.

Não importa qual construtora irá construí-la, desde que esta construtora se enquadre nas regras do contrato. Assim podemos crer que a casa será construída como foi predeterminada. Na prática podemos criar uma *interface* em nosso modelo que terá os métodos que julgamos necessários para determinada ação e desta maneira toda a classe que implementar esta *interface* terá estes métodos e os implementará de acordo com a sua necessidade.

A vantagem é que uma classe pode implementar mais de uma *interface* e com isso manter um relacionamento "é um" com mais de uma estrutura. Observe na **Listagem 1** como utilizamos *interfaces* na sintaxe do PHP.

O simples fato de implementar a *interface IDAO* em nossas classes nos obriga a declarar o método *Gravar()* em todas as classes. Veja que foi declarado um "contrato" (interface), que possui uma única "cláusula" (método), que é *Gravar()*. Agora toda classe que implementar esta *interface* deve respeitar esta cláusula, ou seja, deve implementar este método para fazer conforme sua necessidade. Feito isto criamos duas classes que implementam esta *interface* e cada uma destas classes chamadas concretas implementa o método a sua maneira.

O segredo do código está no método *GravarObjeto*. Observe que ele verifica o parâmetro passado para certificar-se de que ele implementa a *interface IDAO*, ou

seja, ele verifica se a classe passada como parâmetro respeita o contrato. Se a classe segue o contrato então ela terá o método *Gravar()*, pois esta é a única "cláusula" deste contrato. Assim invocamos o método *Gravar()* sem nem mesmo saber de qual classe se trata no momento.

Com *interfaces* resolvemos um problema comum do uso da herança: o acoplamento. Desta maneira programamos para uma *interface* e não para uma implementação. Diminuímos o acoplamento e aumentamos a abstração.

Padrões no PHP

Antes de abordarmos os padrões no PHP é preciso deixar claro que *Design Patterns* não é uma nova linguagem e sim boas práticas no desenvolvimento de software. Estas práticas não são proprietárias desta ou daquela linguagem de desenvolvimento. Já ouvi de algumas pessoas que padrões de projetos eram restritos a linguagem Java, outros dizendo ser impossível aplicar *Design Patterns* em projetos desenvolvidos em Delphi, o que não é verdade. Padrões de projetos podem e devem ser utilizados em toda linguagem que seja Orientada a Objetos. Não será possível aqui abordar todos os padrões descritos pelo *GOF*, porém abordaremos um dos padrões que é frequentemente utilizado: o *Singleton*.

Singleton

Singleton é um padrão *criacional* que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto (**Figura 6**).

Há situações no desenvolvimento de software que queremos garantir que haverá uma e apenas uma instância de determinada classe em nosso projeto. Um exemplo que ilustra bem este cenário seria a classe usuário de uma aplicação. Quantos objetos *Usuário* terão ou precisamos ter instanciados por vez na aplicação? Ou quantos objetos de conexão com o banco de dados precisamos ter? Apenas um. Para estas situações o padrão *Singleton* se aplica perfeitamente.

Abra o Delphi for PHP e crie uma nova aplicação usando *File|New>Application*. Salve-a numa pasta de sua preferência com

Listagem 2. Classe usuario com padrão Singleton

```
<?php
  class Usuario{
    private static $instance;
    public $nome = null;
    private function __construct(){
    }
    public static function GetInstance(){
      if (!isset(self::$instance)){
        self::$instance = new Usuario();
      }
      return self::$instance;
    }
    public function Login($user, $pwd){
      if (($user == 'admin') && ($pwd == 'admin')){
        $this->nome = $user;
        return(true);
      }else{
?>
  <script language="javascript">
    alert('Usuario ou Senha Invalidos');
  </script>
<?php
      }
    }
  }
?>
```

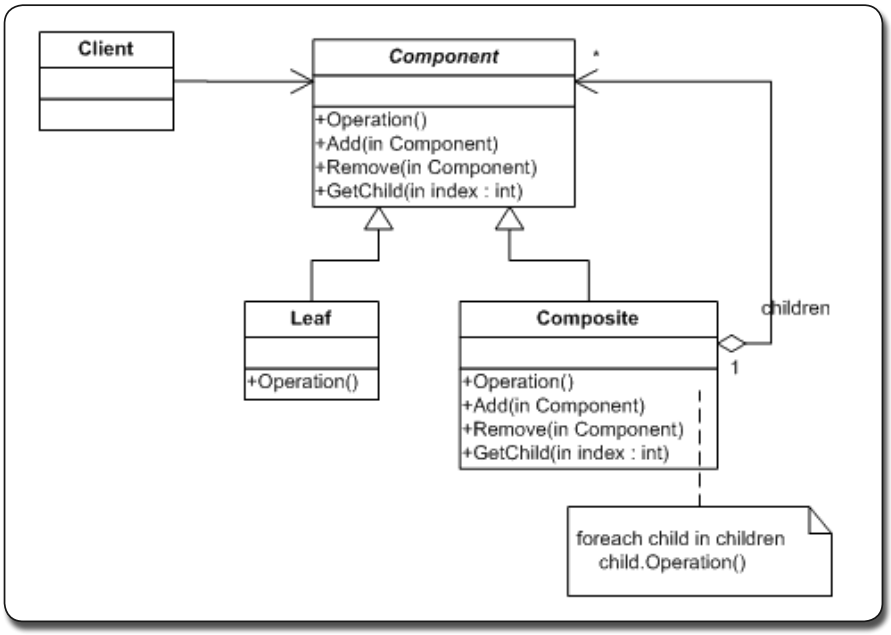


Figura 4. Composite

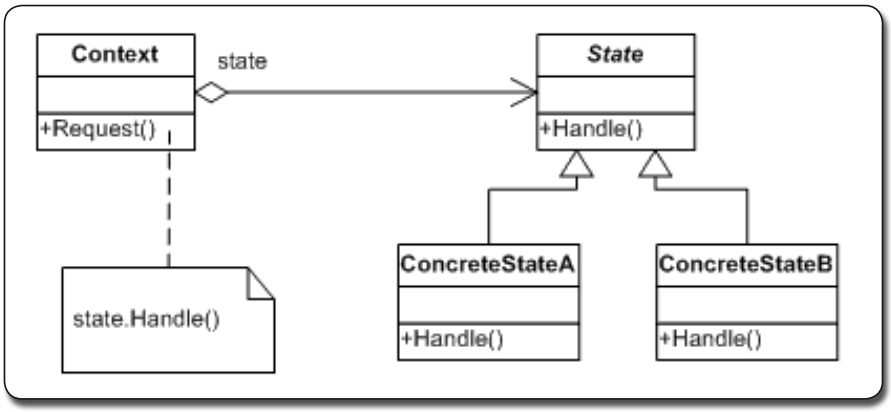


Figura 5. Diagrama UML para padrão State

o nome "Singleton.phprj", e o formulário principal como "index.php". A idéia é criar um sistema de *Login* onde teremos uma classe usuário implementando o *Pattern Singleton*. No formulário principal da aplicação crie uma tela de *Login* conforme a **Figura 7**. Para isso inclua um componente *Panel* da paleta *Standard* e limpe sua propriedade *Caption*. Modifique sua propriedade *BorderColor* usando o valor hexadecimal "#00C6C6" para mudar a cor e o valor *2* na em *BorderWidth* para que tenhamos uma borda mais grossa. Em seguida inclua um *Label* dentro do *Panel* modificando as propriedades *Caption* igual "Acesso Restrito", *Color* igual a "#00C6C6", em *Height* coloque *17*. Modifique também *Font.Case* para *caCapitalize*, *Font.Color* para "#FFFFFF", *Font.Size* igual a "15px" e *Font.Weight* com o valor *bold*. Por fim alinhe manualmente esse *Label* de forma que fique "grudado" com a parte superior do *Panel*. Agora complemente o *Panel* inserindo componentes *Label*, *Edit* e um *Button* assim como na **Figura 7**.

Altere os nomes dos *Edit's* para "EdtUsuario" e "EdtSenha". Adicione ao projeto uma nova *Unit* onde iremos criar nossa classe *Usuário*. Para adicionar essa *Unit* use o menu *File|New>Unit* . Salve-a como "usuario.php". Feito isso nela adicione o código da **Listagem 2**.

A definição deste padrão diz que temos que garantir que apenas um objeto seja instanciado em nosso modelo. Este é o motivo pelo qual se define neste padrão o construtor da classe como privado como segue:

```
private function __construct(){}
```

De outra maneira poderíamos instanciar quantos objetos quiséssemos. Se tentarmos instanciar um objeto através do método *new* uma exceção seria levantada semelhante a mensagem a seguir:

```
Fatal error: Call to private Usuário::__
construct() from context 'Unit2' in
<caminho>\index.php on line 21.
```

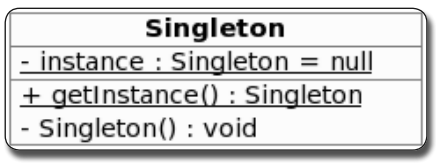


Figura 6. Padrão Singleton

Figura 7. Tela exemplo de login

A definição ainda diz que neste padrão devemos ter um ponto de acesso global. Este ponto é o método estático *GetInstance*. Ele é estático, pois assim não precisamos instanciar a classe para acessá-lo e é nele que fazemos a verificação para saber se já existe uma instância deste objeto em memória. Isto é feito testando-se a variável também estática *$instance*.

Se seu conteúdo for *null* instanciamos um objeto do tipo *Usuario* e guardamos sua referência na variável *$instance*. Do contrário apenas retornamos o ponteiro para o objeto que já existe.

Para verificar o funcionamento do nosso código criamos um método *Login* que recebe dois parâmetros: o *usuário* e a *senha* de quem pretende se logar. Claro que aqui temos um *Login* hipotético, mas nada impede que neste momento seja feito um *Select* ao banco de dados para verificar a existência ou não de um usuário com estes parâmetros. Nesse caso, fixamos o usuário e a senha como sendo *admin*, assim como podemos ver a seguir:

```
...
if (($user == 'admin') && ($pwd == 'admin')){
...
```

Após se logar com sucesso, nós guardamos o nome do usuário logado na propriedade *$nome* e retornamos *true* no método. Caso contrário enviamos uma mensagem ao usuário utilizando *JavaScript*.

Vamos agora à implementação efetiva do nosso exemplo. Ainda na tela de *Login* dê um duplo clique no botão e digite o código a seguir:

```
if (Usuario::GetInstance()->Login($this-
>EdtUsuario->
  Text,$this->EdtSenha->Text)){
    $this->Label3->Caption = "Bem Vindo " .
    Usuario::GetInstance()->nome;
}
```

Observe que não houve a necessidade de intanciar o objeto através do operador *new* e não poderíamos. Como o método *GetInstance()* foi declarado como estático, podemos invocá-lo atrás da classe. Este método irá nos retornar uma instância do objeto usuário e mais do que isto, cada chamada a este método sempre retornará o mesmo usuário.

O que estamos fazendo é bem simples, pedimos uma instância de usuário e invocamos o método *Login* passando o conteúdo dos *Edit's*. Uma vez auten-

Nota do DevMan

Se consultarmos o verbete JavaScript no wikipedia.org veremos a seguinte definição sobre JavaScript:

JavaScript é uma linguagem de programação criada pela Netscape em 1995, que a princípio se chamava LiveScript, para atender, principalmente, às seguintes necessidades:

* Validação de formulários no lado cliente (programa navegador);

* Interação com a página. Assim, foi feita como uma linguagem de script. Javascript tem sintaxe semelhante à do Java, mas é totalmente diferente no conceito e no uso.

1. Oferece tipagem dinâmica - tipos de variáveis não são definidos;
2. É interpretada, ao invés de compilada;\
3. Possui ótimas ferramentas padrão para listagens (como as linguagens de script, de modo geral);
4. Oferece bom suporte a expressões regulares (característica também comum a linguagens de script).

Sua união com o CSS é conhecida como DHTML. Usando o Javascript, é possível modificar dinamicamente os estilos dos elementos da página em HTML.

Dada sua enorme versatilidade e utilidade ao lidar com ambientes em árvore (como um documento HTML), foi criado a partir desta linguagem um padrão ECMA, o ECMA-262, também conhecido como ECMAScript. Este padrão é seguido, por exemplo, pela linguagem ActionScript da Adobe (antiga Macromedia).

Além de uso em navegadores processando páginas HTML dinâmicas, o JavaScript é hoje usado também na construção do navegador Mozilla, o qual oferece para a criação de sistemas GUI todo um conjunto de ferramentas (em sua versão normal como navegador, sem a necessidade de nenhum software adicional), que incluem (e não apenas) um interpretador de Javascript, um comunicador Javascript <-> C++ e um interpretador de XUL, linguagem criada para definir a interface gráfica de aplicações.

O uso de JavaScript em páginas XHTML, pelo padrão W3C, deve ser informado ao navegador da seguinte forma:

```
<script type="text/javascript">
/* aqui fica o script */
</script>
```

Em outras palavras, o JavaScript é uma linguagem de script que nos auxilia no desenvolvimento de aplicações para Web, sejam ela desenvolvidas em PHP, ASP.NET, HTML, ou qualquer outra linguagem destinada a estes fins.

ticado, apenas exibimos no *Label* uma mensagem de boas-vindas ao usuário. Mas como saber se realmente só há uma instância do objeto? Simples. Repare que para recuperar o nome do usuário também utilizamos o *GetInstance()*. Se um novo objeto fosse retornado, o conteúdo da propriedade nome seria vazio, pois atribuímos o valor a esta propriedade no método *Login*. Assim, caso uma nova instância fosse criada ao chamar o método *GetInstance()* pela segunda vez a mensagem exibida seria apenas "Bem-Vindo". Execute a aplicação, faça o *Login* e observe o resultado na **Figura 8**.

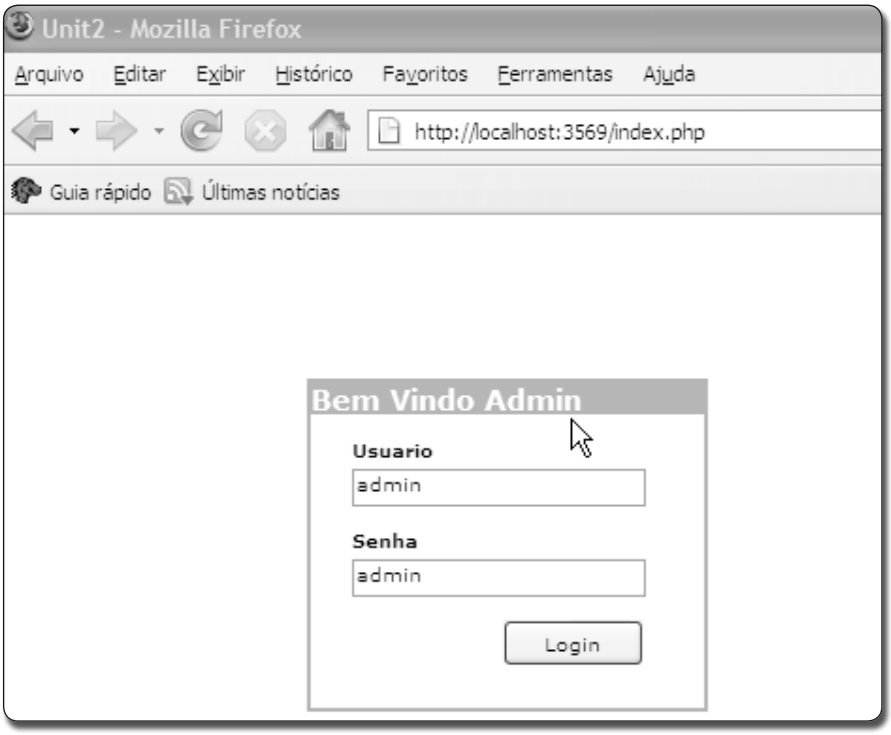


Figura 8. Aplicação em Execução

Conclusão

Concluímos assim esta pequena introdução sobre o mundo OO no Delphi for PHP. Muito ainda há o que se falar, principalmente sobre padrões de projeto. Lembrando que orientação a objetos gera muita discussão por ser um tema polêmico e altamente detalhado. Recomendo a consulta de livros e comunidades especializadas no assunto, pois quanto maior for o aprofundamento das técnicas de OO, melhor êxito terá no desenvolvimento de suas aplicações futuras.

Um abraço a todos e até a próxima. Eu sou Rodrigo Carreiro e pela sua atenção, muito obrigado! ●

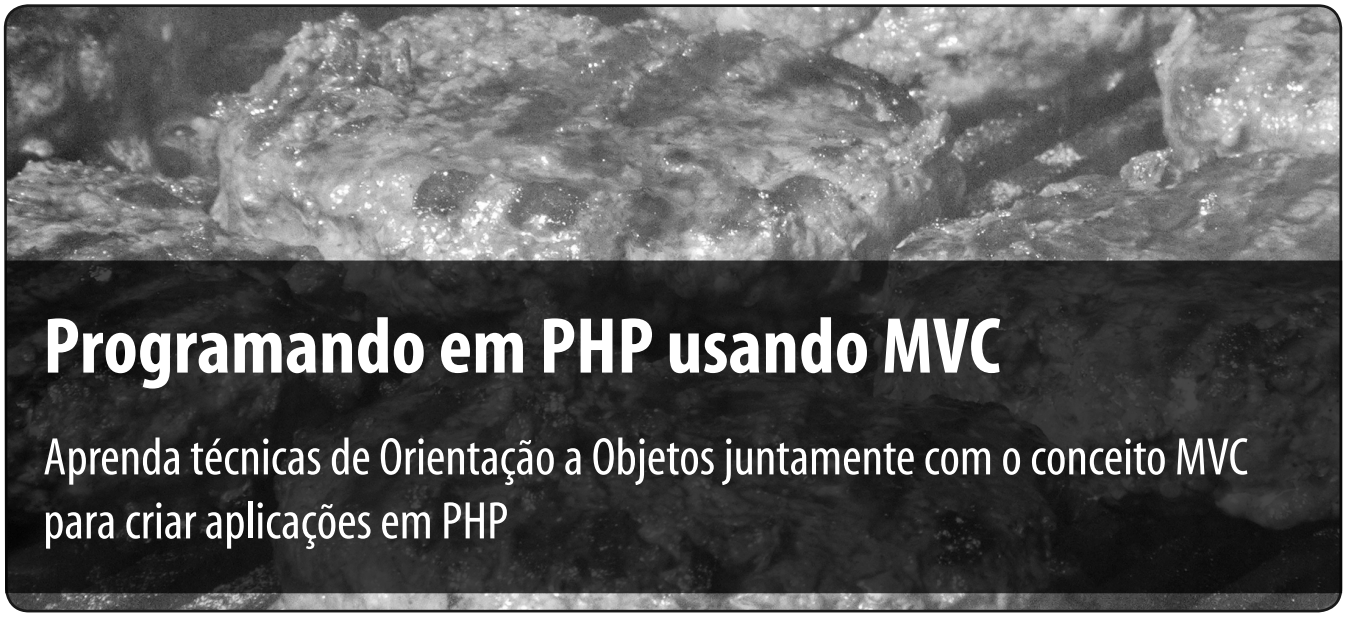
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP



Programando em PHP usando MVC

Aprenda técnicas de Orientação a Objetos juntamente com o conceito MVC para criar aplicações em PHP

Daniel Ribeiro
(daniel@danielribeiro.com)
analista de Sistemas e especialista em desenvolvimento avançado em PHP há 6 anos. Bacharel em Ciências da computação pela Universidade do Grande ABC. Desenvolvedor e coordenador de projetos do grupo Sonda Procwork.

Muita gente já ouviu falar alguma vez sobre esta metodologia de desenvolvimento. Mas afinal, o que vem a ser uma aplicação desenvolvida em MVC? Neste artigo, irei explicar de uma forma sucinta sobre este tipo de arquitetura de desenvolvimento e como este método pode ser aplicado utilizando a linguagem PHP.

MVC vem da sigla (Model-View-Controller) e é um padrão de arquitetura de software. Em aplicações complexas, que enviam uma série de dados para o usuário, o desenvolvedor frequentemente necessita separar os dados ("Model") da interface ("View"). Desta forma, alterações feitas na interface não afetarão a manipulação dos dados, e estes poderão ser reorganizados sem alterar a interface do usuário. O *model-view-controller* resolve este problema através da separação das tarefas de acesso aos dados e lógica do negócio da apresentação e interação com o usuário, introduzindo um componente entre os dois: o *Controller*. MVC é usado em padrões de projeto de software, mas MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

A **Figura 1** representa um diagrama simples exemplificando a relação entre *Model*, *View* e *Controller* onde linhas sólidas indicam associação direta e as tracejadas indicam associação indireta. É comum dividir a aplicação em camadas separadas: apresentação ("interface"), domínio e acesso a dados. Em MVC a camada de apresentação também é separada da *view* e da *controller*.

MVC é muito visto também em aplicações para Web, onde a *View* é geralmente a página HTML, e o código que gera os dados dinâmicos para dentro do HTML é o *Controller*. E, por fim, o *Model* é representado pelo conteúdo de fato, geralmente armazenado em bancos de dados ou arquivos XML.

Usando Objetos para entender o conceito aplicado de MVC

Muita gente me pergunta sobre a maneira que eu programo usando MVC com PHP. É claro que já existem vários *frameworks* que utilizam essa técnica no desenvolvimento PHP. Porém resolvi apresentar aqui o conceito do que uso que pode servir de material de auxílio para outros programadores também.

Para se ter uma idéia, vamos partir para o modelo de programação orientada a objetos. A maneira mais prática de se programar usando MVC nada mais é do que programar usando orientação a objetos. Vamos pensar então no conceito de objeto. Um exemplo, um veículo, onde teremos a classe de veículo e seus métodos (**Listagem 1**).

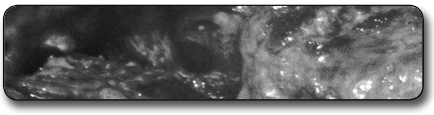
Agora digamos que temos vários veículos, tais como: carros, motos, bicicletas e cada um terá eventos diferentes um do outro. Para isso criamos novas classes estendendo a classe *Veiculo* no qual as novas classes irão herdar os métodos e propriedades da classe Veiculo, como podemos ver a seguir na **Listagem 2**.

Desta forma quando instanciarmos o objeto *Bicicleta*, podemos utilizar os métodos de *Veiculo*, pois *Bicicleta* é uma extensão de *Veiculo*, como no código seguinte:

```
<?php
  $meuVeiculo = new Bicicleta();
  $meuVeiculo->trocarMarcha();
  $meuVeiculo->andar();
  $meuVeiculo->parar();
?>
```

Agora vamos construir outra classe *Automovel* que instancia *Veiculo* (**Listagem 3**). Desta forma quando instanciarmos o objeto *Automovel*, podemos utilizar os métodos de *Veiculo*, pois *Automovel* é uma extensão de *Veiculo*, como podemos na **Listagem 4**.

Percebeu que desta forma, se quisermos mudar o comportamento de um veículo (independente do que ele seja) basta modificar a classe, que todos os tipos de veículos serão alterados (pois eles estendem a classe *Veiculo*). Isso facilita muito na manutenção do código.



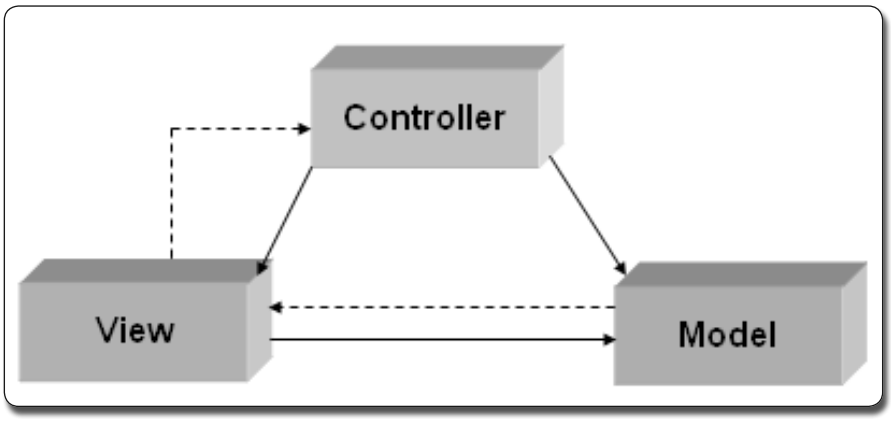


Figura 1. Representação gráfica do conceito MVC

Listagem 1. Exemplo de classe em PHP

```
<?php
class Veiculo{

  /* Qualquer veículo tem os seus métodos padrões: */
  public function andar(){

  /* Programação da função */
  }
  public function parar(){

  /* Programação da função */
  }
}
?>
```

Listagem 2. Classe veículo estendida

```
<?php
class Bicicleta extends Veiculo{

  /* A bicicleta por sua vez contem propriedades */
  public $rodas;

  /* E tbm tem um metodo construtor */
  public function __construct(){
    $this->rodas = 2;
  }

  /* E tbm seus métodos */
  public function trocarMarcha(){

  /* Programação da função */
  }
}
?>
```

Listagem 3. Classe Automóvel que vem de Veículo

```
<?php
class Automovel extends Veiculo{
  public function ligar(){

  /* Programação da função */
  }
  public function desligar(){

  /* Programação da função */
  }
}
?>
```

Listagem 4. Criação do objeto Automovel

```
<?php
  $meuVeiculo = new Automovel();
  $meuVeiculo->ligar();
  $meuVeiculo->andar();
  $meuVeiculo->parar();
  $meuVeiculo->desligar();
?>
```

Criando sua aplicação em MVC

Pois bem, onde queremos chegar com tudo isso? Uma aplicação MVC nada mais é do que classes principais de:

• *(M)odel* - Utilizada como o negócio da sua aplicação, (tudo o que for utilizado para armazenamento de dados ou obtenção dos dados da aplicação);

• *(V)iew* - Utilizada para exibir para o usuário tudo o que a aplicação produz;

• *(C)ontroller* - Utilizada para realizar o controle integrando o *Model* e *View* (é aqui e controlamos o fluxo da aplicação e as ações do usuário);

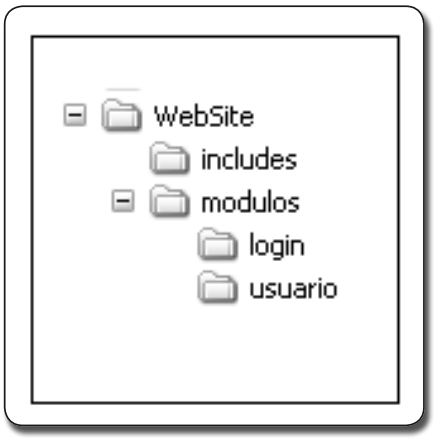


Figura 2. Estrutura de pastas de nosso exemplo

Bem, se sabemos que o *Model* interage com o banco de dados, então criamos uma classe *model* principal com os métodos principais que ele tem, veja na **Listagem 5**. O código dado é somente um mero exemplo, pra você ter uma idéia. É claro que em uma *Model* existem mais métodos para acessar os dados, dependendo da sua aplicação. Crie uma estrutura de pastas para seu site como mostrado na **Figura 2**. Em seguida abra o Bloco de Notas ou outro editor de textos a sua escolha e digite o código da **Listagem 5**. Salve esse arquivo dentro da pasta *\WebSite\includes* com o nome "Model.php". Agora temos a classe *View* na **Listagem 6**. Fiz questão de comentar todas as listagens para que entendamos melhor cada função, cada declaração e cada implementação.

Se você perceber, o *Smarty* segue um critério semelhante, não sei se você já usou *Smarty*, mas ele funciona como na **Listagem 7**. Já que temos métodos semelhantes, podemos então aplicar em nossa *View* todos os métodos do *Smarty*, para não ter que se preocupar em trabalhar com manipulação de *templates*. Veja como ficaria nossa *View* baseando-se em *Smarty* na **Listagem 8**. Digite o código da **Listagem 8** em um novo arquivo e salve-o como "View.php".

Talvez você esteja se perguntando: "Se vou usar o *Smarty*, por que então tenho que criar uma *View* e usar os métodos do *Smarty* dentro da *View*? Não poderia usar diretamente o *Smarty*?"

Poderia, mas digamos que você posteriormente não queira mais utilizar o *Smarty*, e sim outro gerenciador de *templates*. Você teria nesse caso que mexer em sua aplicação inteira? Fazendo uma classe dessa forma, você só mexe na *View*, e programa *Ela* pra usar outro gerenciador de *templates*.

Passamos para a classe *Controller*, que contém os métodos básicos que farão a sua aplicação funcionar. Até mesmo métodos genéricos que você utiliza dentro da sua aplicação. Veja o código da **Listagem 9**. Novamente crie um novo arquivo e digite o código da **Listagem 9** que encontra-se comentado.

Resumindo, a classe *Controller* irá utilizar métodos que são genéricos para todos os módulos de sua aplicação. Agora que já temos o *Model*, *View* e *Controller*, vamos criar os módulos de nossa aplicação usando MVC.

Listagem 5. Criação da classe Model

```
<?php
class Model{
  public function conectar(){
    /* Código pra conectar-se ao banco de dados */
    $conexao = mysql_connect("localhost", "database", "senha");
    mysql_select_db("meu_database");
  }
  public function query($tabela){
    /* Código pra executar uma query */
    $sql = "SELECT * FROM " . $tabela;
    $query = mysql_query($sql);
  }
  public function desconectar(){
    /* Código pra desconectar do banco */
    mysql_close($conexao);
  }
}
?>
```

Listagem 6. Criação da classe View

```
<?php
class View{
  public function mostrarNaTela($template){
    /*
      Mostra na tela usando o template escolhido,
      aqui vem o código pra ele dar include em
      algum arquivo html modificando os valores
      que já estão atribuidos a View etc...
    */
  }
  public function atribuirValor($var, $valor){
    /* Código pra atribuir valores na view */
  }
}
?>
```

Nota do DevMan

O que é Smarty?

O Smarty é uma classe de templates. Funciona de uma forma que separe interface da lógica de programação e tem por objetivo, facilitar e melhorar o desenvolvimento de qualquer aplicação em PHP.

Por ser muito difundido no mundo inteiro, e estar ligado ao site oficial do PHP, o Smarty tem uma comunidade grande de desenvolvedores. Isso ajuda no suporte e discussão de melhorias.

A sua última versão é a 2.6.11, lançada em Dezembro de 2005.

Principais funcionalidades

* Melhor desempenho de execução;
* Acaba com overhead de template, pois compila apenas uma vez;
* Você pode criar suas próprias funções, é extensível;
* Você configura a sintaxe da tag de delimitação do template. Assim pode-se usar {}, {{}}, <!--{}-->, etc;
* Uso de funções condicionais de forma simplificada dentro dos templates;
* Sem limitação para loops, ifs, etc;
* Modo de cache embutido;
* Arquitetura de plugins.

Criando módulos estendendo suas classes MVC

Um dos módulos é o de *login*, então teremos o *loginModel.php*, *loginView.php* e *loginController.php*. Cada um desses serão classes que estenderão suas respectivas "classes mãe". Agora sim iremos para a classe do controlador, que irá interagir com *loginModel.pnp* e *loginView.php*. Veja o código de cada classe nas **Listagens 10, 11** e **12**. O que faremos é separar cada Listagem em um arquivo a parte, como pode notar. Portanto **Listagem 10** será *loginModel.php*, **Listagem 11** igual a *loginView.php* e *loginController.php* conterá a **Listagem 12**. Todos esses arquivos deverão ser salvos na pasta *\WebSite\modulos\login*. Note que no início de cada script temos uma chamada à função *include* do PHP. Isso significa que estamos incluindo o código dos arquivos *Model.php, Controller.php* e *View.php* que possuem as definições das *classes mãe*.

> **Nota**: Para fins didáticos, simplificamos ao máximo a estrutura das classes e também por estarem fora do escopo desse artigo, alguns métodos não foram implementados, tais como *gravaSessao*, *apagaSessao* e *log*.

Agora que já temos o nosso controlador *loginController.php*, temos que fazer com que ele seja instanciado e chamado pelo usuário, para isso teremos um arquivo de *login* onde irá fazer a instância do nosso controlador. Algo como "login.php", este arquivo será chamado pelo navegador, onde fará a requisição da sua classe.

Requisitando os módulos em sua aplicação

Pensando da maneira como foi descrito anteriormente, você terá que fazer um arquivo para cada módulo do seu sistema, se o seu sistema tiver vários módulos, então terá vários arquivos:

- login.php;
- usuarios.php (módulo de controle de usuários);
- produtos.php (módulo de controle de produtos);

Se formos ver, todos os arquivos que fazem as requisições de suas classes

Listagem 7. Aplicação do Smarty

```
<?php
  $smarty = new Smarty();
  /* Em nossa View o método atribuirValor() */
  $smarty->assign('var1', $var1);
  $smarty->assign('var2', $var1);
  /* Em nossa View o método mostrarNaTela() */
  $smarty->display('template.tpl');
?>
```

Listagem 8. View baseada em Smarty

```
<?php
class View{
  /*
    Declaramos a propriedae Smarty aqui dentroda nossa View (veja que é uma propriedade
    private, pois somente a classe View pode acessá-la.
  */
  private $smarty;
  public function __construct(){
    /*
      Quando então for chamado o contrutor da classe, ele instancia o objeto Smarty.
    */
    $this->smarty = new Smarty();
  }
  public function mostrarNaTela($template){
    /*
      Chama o método do smarty pra exibir o template na tela.
    */
    $this->smarty->display($template);
  }
  public function atribuirValor($var, $valor){
    /*
      Chama o metodo do smarty para atribuir o valor a variável.
    */
    $this->smarty->assign($var, $valor);
  }
}
?>
```

Listagem 9. Código de criação do Controller

```
<?php
class Controller{
  public function redirect($url) {
    header('Location: ' . $url);
  }
  public function error($error){
    /*
      Se algum erro acontecer, chamamos esse método, que vai fazer alguma coisa (mandar
      um email,gravar log, etc
    */
    $this->log($error);
  }

  public function log($mensagem){
    /*
      Grava uma menagem de log em algum arquivo qualquer definido aqui
    */
    }
}
?>
```

Listagem 10. Criação da classe loginModel herdada de Model

```
<?php
require_once('includes/Model.php');
class loginModel extends Model{
  /*
   * Método para verficiar a senha do usuario
   * @param string $usuario
   * @param string $senha
   * @return bool
  */
  public function verificarSenhaUsuario($usuario,
    $senha){
    $this->conectar();
    $sql = "SELECT * FROM usuarios WHERE usuario =
      $usuario AND senha = $senha";
    $resultado = $this->query($sql);
    $this->desconectar();
    return $resultado;
  }
}
?>
```

Listagem 11. View da classe Login

```
<?php
require_once('includes/View.php');
class loginView extends View{
  /* Exibe a tela de login */
    public function exibirTelaLogin(){
      $this->mostrarNaTela('login.tpl');
    }
  /* Exibe a tela de erro */
  public function exibirTelaErro(){
    $this->mostrarNaTela('erro.tpl');
  }
  public function exibirTelaLogado(){
    $this->mostrarNaTela('logado.tpl');
  }
}
?>
```

Listagem 12. Controller do Login

```
<?php
class loginController extends Controller{
  /*
    O loginController vai agregar loginModel e loginView, então criamos esses objetos aqui
    * @var loginModel
  */
  private $model;
  /* @var loginView */
  private $view;
  public function __construct(){
    /* Instanciamos os objetos */
    $this->model = new loginModel();
    $this->view = new loginView();
  }
  public function telaLogin(){
    /* O método contrutor ja chama a view do login pra exibir a tela de login */
    $this->view->exibirTelaLogin();
  }
  public function fazerLogin(){
    /* Tenta ver se o usuario colocou a senha certa */
    if ($this->model->verificarSenhaUsuario(
      $_POST['usuario'], $_POST['senha'])){
      $this->gravaSessao();
      $this->log('usuario logou');
      $this->view->exibirTelaLogado();
    }else{
      $this->log('Usuário tentou logar mas não conseguiu');
      $this->view->exibirTelaErro();
    }
  }
  /* Esse método é private, significando que só pode ser chamado dentro da classe */
  private function gravaSessao(){
    /* Grava a sessão usando session ou cookie */
  }
  /* Esse método é private, significando que só podem ser chamando dentro da classe */
  private function apagaSessao(){
    /* Limpa a sessao ou cookie */
  }
}
?>
```

Listagem 13. Código do arquivo login.php

```
<?php
  $objLogin = new loginController();
  /*
    Verificamos que método o browser está chamando.
    Se não chamar nenhum método, exibimos a tela delogin. Desta forma se for chamado a
    url http://www.seusite.com.br/login.php ele vai chamar o método padrão telaLogin
  */
  if (!$_REQUEST['action']){
    $_REQUEST['action'] = 'telaLogin';
  }
  /* Agora executamos o método passado via url */
  eval('$objLogin->' . $_REQUEST['action']);
?>
```

Listagem 14. Código do arquivo main.php com chamadas generalizadas aos módulos

```
<?php
  /* Instância o módulo */
  eval('$instancia = new ' . $_REQUEST['modulo'] . 'Controller();');

  /* Define uma acao default */
  if (!$_REQUEST['action']) {
    /* Isso implica que todos os controllers terão que ter um método chamado acaoPadrao */
    $_REQUEST['action'] = 'acaoPadrao';
  }
  /* Agora eu executo o método passado via url */
  eval('$instancia' . $_REQUEST['action']);
?>
```

são semelhantes, então por que não generalizar e fazer em um arquivo? Um *script* que pode ser usado para chamar qualquer módulo do sistema? Mãos à obra. Crie um novo arquivo PHP, dessa vez na pasta raiz de nossa aplicação, e salve-o como "main.php". Nele digite o código da **Listagem 14**. Agora você tem um arquivo que pode chamar qualquer módulo/método do seu sistema, bastando especificar na url.

```
http://www.seusite.com.br/main.
php?module=login
http://www.seusite.com.br/main.
php?module=usuario
http://www.seusite.com.br/main.
php?module=produtos
```

Conclusão

O que escrevi aqui tem como principal objetivo mostrar, de forma didática, como construir aplicações usando o modelo MVC. É claro que depois de entender perfeitamente o conceito, tudo fica mais fácil e torna-se perfeitamente possível aperfeiçoar ainda mais sua aplicação personalizando, se necessário, do jeito que melhor se adapte às necessidades de seus projetos.

Existem muitos *frameworks* que auxiliam na codificação de suas aplicações usando a arquitetura MVC. Para mais referências veja alguns deles na seção Links. ●

Links

CakePHP - Framework MVC para PHP
http://www.cakephp.org/

CodeIgniter - Framework MVC para PHP
http://www.codeigniter.com/

Framework PHP.MVC - Framework em PHP no padrão MVC
http://www.phpmvc.net/

Prado - Framework MVC para PHP
http://www.xisc.com/

Symfony - Framework MVC para PHP
http://www.symfony-project.com/

Zend Framework - Framework em PHP no padrão MVC
http://framework.zend.com/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** d site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia

www.devmedia.com.br/creditos

Microsoft

ROMA LEVOU MIL ANOS PARA SER CONSTRUÍDA.
SUA EQUIPE TEM UM MÊS.

ENFRENTE OS DESAFIOS

Desafio: terminar projetos com qualidade dentro dos prazos. Estratégia: mais controle e previsibilidade no processo de desenvolvimento com o Visual Studio® Team System. Veja dicas e ferramentas em **enfrenteosdesafios.com.br**

Microsoft® Visual Studio

©2008 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Studio, o logo do Visual Studio e "Seu potencial. Nossa inspiração" são marcas comerciais, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou em outros países. Os nomes